ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Integrated Systems Laboratory
Prof. L. Benini
Spring Semester 2019

Group Project

Demonstrator: A mobile and user-friendly power analyzer

Authors:
Jonas Künzli
René Zurbrügg
Marina Homs

Superivsors:
Philipp Mayer
Manuel Eggimann
Michele Magno

Contents

List of Figures

Abstract

During this group project we designed, assembled and tested a so-called Demonstrator, which can measure currents as well as voltages. The measured data is then nicely displayed on a touch screen. Additionally, the device can generate different supply voltages up to 3.5 V very precisely, which can then be used for different applications (see chapter 6).

Chapter 1

Introduction and Motivation

PULP stands for parallel ultra low power and is a project of the Integrated Systems Laboratory (IIS) of the ETH Zurich as well as of the Energy-efficient Embedded Systems (EEES) group of UNIBO. The main goal of PULP is to establish an open and scalable hardware and software research platform which is very energy efficient.

To demonstrate the characteristics of these low power devices a tablet was required which could measure the voltages and currents of any PULP chip and display them nicely on a monitor. On the other hand the device should also be able to generate different supply voltages that are needed depending on the PULP chip you want to measure data from. In this group project we developed such a device or how we call it a DEMONSTRATOR.

The main purpose of the DEMONSTRATOR is not to replace an oscilloscope or a precision measurement device but to measure currents and voltages of any chip quite accurately and especially to display the measurements appealing on the display. The device is very handy and runs with a battery. We also included a JTAG Programmer in our device, which you can program a connected PULP chip with. Thanks to those features the DEMONSTRATOR serves mainly as demonstration purposes. For example you can easily take the device with you to a conference and everything you need for a live demonstration of a PULP chip is offered by the DEMONSTRATOR. Or other useful applications of the device are that you can do outdoor measurements, where no electricity is present, you can do real time measurements and program a connected PULP chip steadily.

Chapter 2

Set up of the Demonstrator

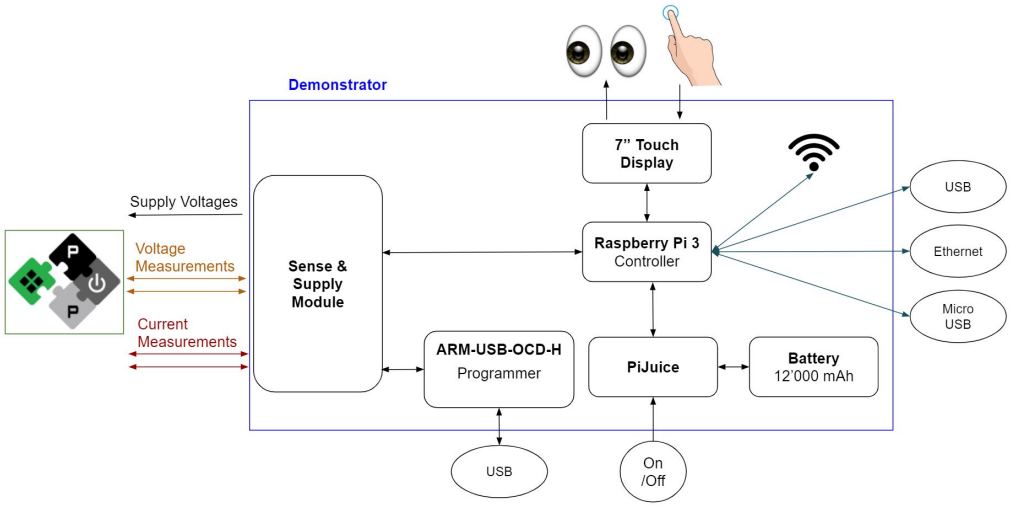


Figure 2.1: The Set Up of the Pulp Demonstrator

As one can see in the figure 2.1, a central component of the Demonstrator is the Sense & Supply Module we designed by ourselves during the group project. On one hand it is responsible for all the current and voltage measurements. There are two channels for each measurement type which could be used simultaneously. The measurements are very accurate (see chapter 7.4) and depending on the application one can chose between different resolution ranges (for the voltage measurement there are 5 V, 500 mV and 50 mV ranges and for the current measurements there are 50 mA and 500 µA ranges). On the other hand the module can generate different supply voltages up to 3.5 V with different accuracy depending on the application it is used for.
To access those different supply voltages there is a 2x20 pin header located on

one side of the Demonstrator, where one can get the required voltages and connect them to any desired application. Similarly, access to the current and voltage measurements is granted by connecting whatever one would like to measure to the according pins. Another central part of the Demonstrator is a Raspberry Pi. It communicates with all the components that make up the Demonstrator and controls them. For example, by a I^2C connection the Raspberry Pi can set the pins of any IC to low or high so that the measurement is done correctly. On the integrated SD card the booting process is defined and the Raspberry Pi knows how to load all the settings or how to monitor the correct site after it has been turned on again (see chapter 4).
The USB or the Ethernet connection of the Raspberry Pi can be used to transfer data easily to or from the Raspberry Pi for different purposes. To work with a computer mouse and keyboard the USB connection is also very useful. The Wifi hotspot of the Raspberry Pi can be used to display the measured data not only on the Demonstrator but also on other devices that can connect to the hotspot (see chapter 4). Thanks to the micro USB connection one can easily charge the device once the battery is empty.
Talking about the battery, there is a battery with a capacity of 12 000 mA h present in the Demonstrator. This means that our device can run approximately for 8 hours without being charged. The battery is connected to a so called PiJuice. The PiJuice is put on top of the Raspberry Pi and communicates via I^2C with it. It is responsible for the power management of the Demonstrator and has all the information of the battery such as the actual charging level and temperature.
To turn on and off the Demonstrator there is a button placed on one side of the device. It is connected to the PiJuice as well.
There is also an Olimex ARM-USB-OCD-H programmer included in the device which can be used to program any connected pulp chip, this might be necessary depending on the pulp chip.
To visualize the measured data a display is required. In this case there is a 7" touch display connected to the Raspberry Pi where the measured data are continuously represented in a plot. Thanks to the monitor the user can also easily change the settings of the device manually. For example all the supply voltages can be set however they are needed at the moment, change the colours of the plots, activate or deactivate the different measurement channels, set the resolution ranges for the measurements and much more.

Chapter 3

Hardware Description

On our self made Sense & Supply Module there are four different pin headers pointing outwards as seen in figure 3.1. There is a header that connects the Sense & Supply Module with the Raspberry Pi, a header for the supply voltages of the touch display, a connector for the Olimex ARM-USB-OCD-H programmer and the main connector for measurements and output supplies.
The PCB itself consists of three main parts. There is a part made up of the internal voltage supplies, a second one to get in touch with the surroundings by different voltage outputs and a third part consisting of the current and voltage measurements.

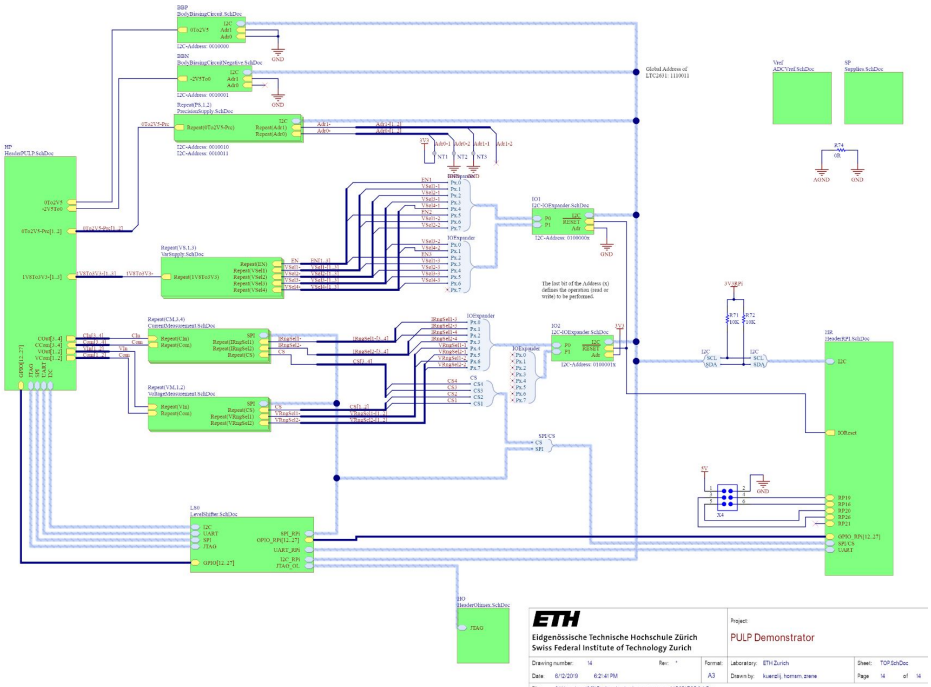


Figure 3.1: Top level of schematic.

3.1 Sense & Supply Module

The main goal of this header is to power an external board with variable power supplies and to measure voltages and currents on the fly.

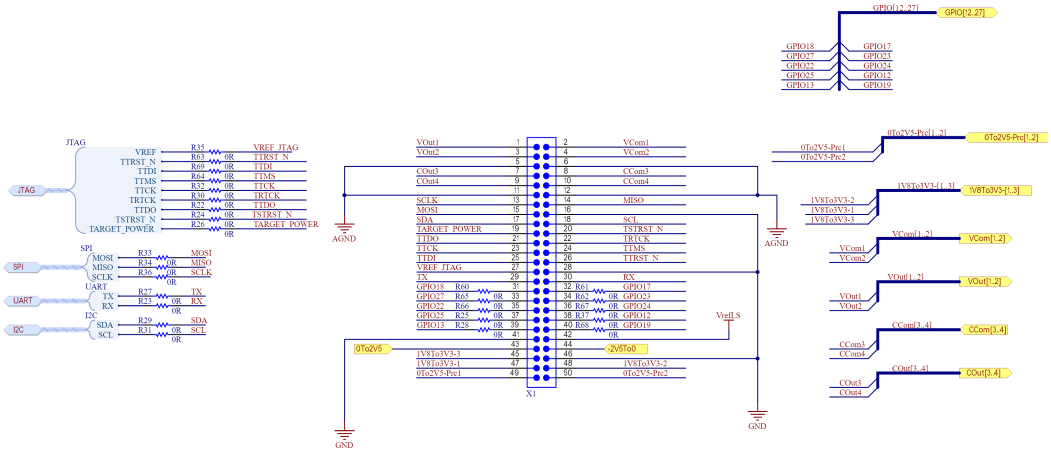


Figure 3.2: Block diagram of the PULP Header.

3.1.1 Power Supplies

To supply an external circuit our board offers three adjustable supplies called Variable Supplies with a range from 1.8 V to 3.3 V and two Precision Supplies with a range from 0 V to 2.5 V. Additionally there are two so called Body Bias. One of them has a range from 0 V to 2.5 V and the other one has a range from −2.5 V to 0 V which can be seen on the right side of the figure 3.2.

3.1.2 Measurements

Our board provides two voltage and two current measurements. All four measurements have their own input pin and common pin. Resolution ranges of 50 mV, 500 mV or 5 V can be chosen for the voltage measurements and for the current measurements there are resolution ranges of 500 µA or 50 mA. The measured values are committed to the Raspberry Pi via SPI.

3.1.3 Buses

On the left side of the figure 3.2 there are four buses shown. On the one hand JTAG is connected through a level shifter to the programmer. On the other hand I^2C, UART and SPI are connected through level shifters to the Raspberry Pi header. With the input pin VRefLS (**V**oltage **Ref**erence **L**evel **S**hifter) the voltage of the level shifter can be set. In our case the maximum voltage to set is 3.3 V.

3.1.4 GPIOs

Different GPIOs of the Raspberry Pi are transmitted to the pulp header as seen in the figure 3.2 in the top right corner. Again the voltage level can be set with VRefLS. Additionally, 0 Ω resistors near the header can be desoldered to decouple the GPIOs from the header if needed.

3.2 Internal Voltage Supplies

The 5 V pins of the Raspberry Pi are needed to power our board, the Sense & Supply Module. The voltage of the Raspberry Pi is converted to the different desired voltages. They are all generated according to the same principle. First, a buck converter breaks down the voltage. After that, a low dropout (LDO) regulator generates a smooth and precise voltage. As seen in figure 3.3 the ICs TLV62565DBVT and TPS62740DSST convert the 5 V to 3.5 V or respectively 2.9 V. These two voltages are 0.2 V higher than the desired voltages of 3.3 V respectively 2.7 V because both LDOs TLV75533PDBVR and TPS78227DDCR have a dropout voltage of approximately 0.2 V at their maximum output current. The TPS60403QDBVRQ1 is additionally connected to the 3.3 V and generates −3.3 V.

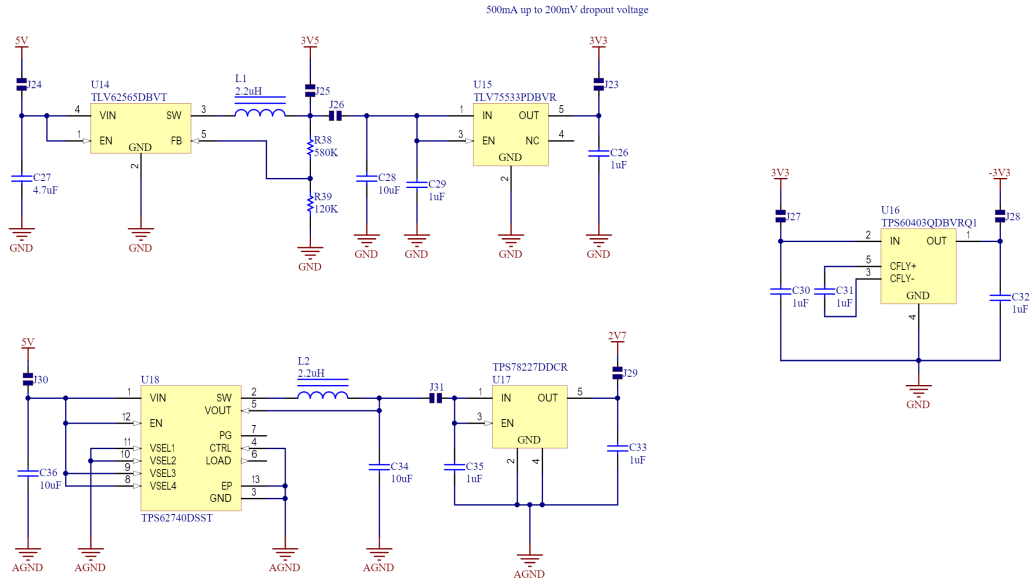


Figure 3.3: Block diagram of internal supplies.

Furthermore, the part at the bottom on the left hand side of the figure 3.3 is coupled with the analog ground because it powers only analog devices. The analog and digital ground are separated and only connected via a 0 Ω resistor.

3.3 Body Bias

The two body biasing circuits consist of LTC2631CTS8-LZ12#TRMPBF as seen on the left side of the figure 3.4. This IC has a full-scale 12-bit output of 2.5 V. The internal reference voltage is available at pin number 6 and is bypassed to GND by a capacitor of 330 nF. The output voltages can be adjusted over I^2C. The LM321MF is added at the output as a voltage follower. The negative body bias circuit is similar. Instead of a voltage follower, the operation amplifier acts as an inverting amplifier with gain of -1. Thus, we can generate negative voltages.

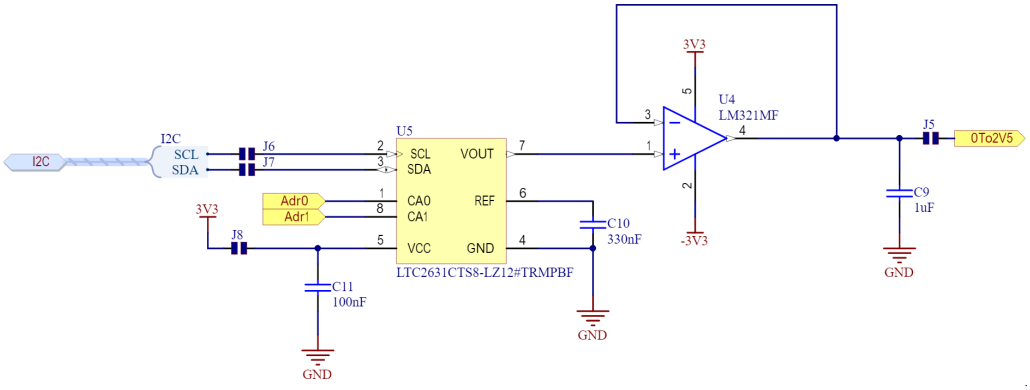


Figure 3.4: Block diagram of body bias.

3.4 Precision Supply

As you can see in figure 3.5, the IC to generate the different voltages is the same as in the body biasing schematic 3.3. Instead of LM321MF the MAX9613AXT+ is used and in addition, the chip has its own supply voltage to ensure a precise voltage.

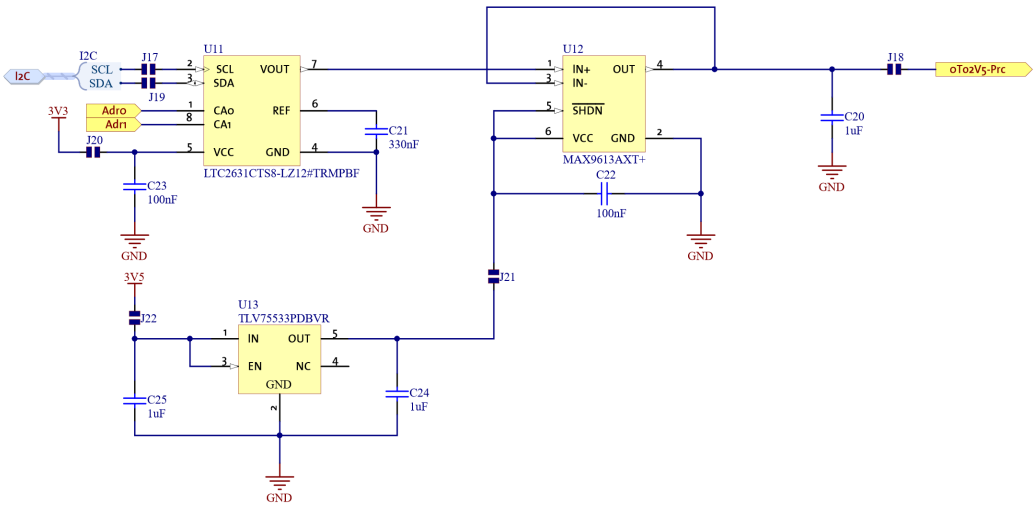


Figure 3.5: Block diagram of precision supplies.

3.5 Variable Supply

For the variable supply we chose the TPS62742DSST as seen in figure 3.6, which supports 16 selectable output voltages in 100 mV steps between 1.8 V to 3.3 V. The voltages can be set using the input pins VSel1 - VSel4. The CTRL pin (pin 4) is pulled to GND, therefore the LOAD pin (pin 6) is disabled.

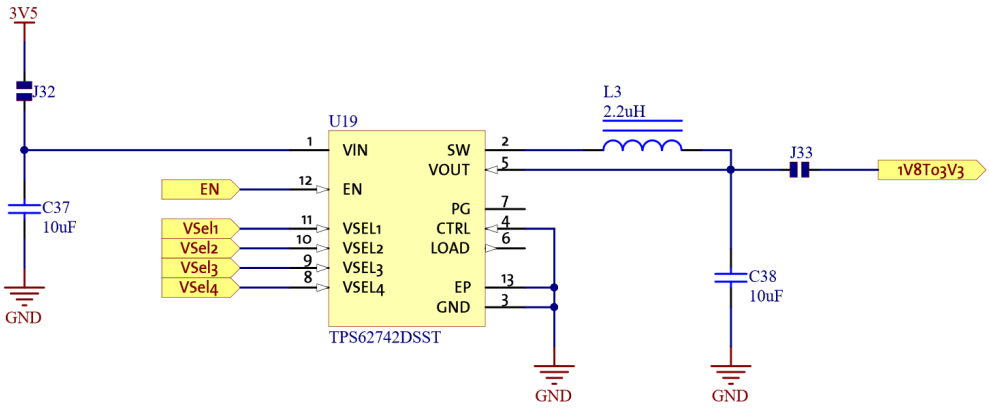


Figure 3.6: Block diagram of variable supplies.

3.6 IO Expander

To select a voltage of the variable supply as described in section 3.5 or to change between the different voltage respectively current ranges as described in section 3.8/3.9, we added the IO expander PCAL6416APW,118 to our board. With I^2C commands it is possible to control the IO Expanders outputs.

3.7 Analog Reference

The analog reference circuit generates a voltage of 2.5 V and 1.35 V. The 2.5 V is connected to the reference pin of the ADS8885, while the 1.35 V supports the common-mode voltage. The common-mode voltage is set to half the supply voltage of the analog part to achieve a maximal dynamic range. We adopted the values of the resistors and capacitors from the schematics of Texas Instruments[1].

3.8 Voltage Measurement

The voltage measurement consists of three parts. In the beginning there is a voltage divider, which can be changed to keep the maximum voltage consistent in the different voltage ranges. After the signal has passed the divider it will be amplified and finally the signal can be measured with the ADS8885IDGSR.

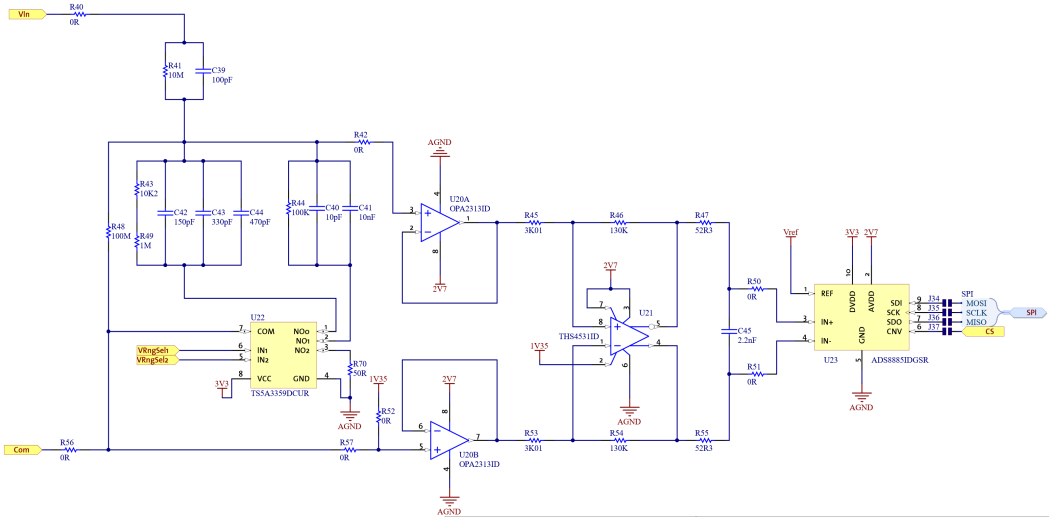


Figure 3.7: Blockdiagram of voltage measurement.

As seen on the left side of the figure 3.7, the voltage divider can be configured in three ways with VRngSel1 and VRngSel2. None or one of the two wires can be connected in parallel to the 100 MΩ resistor. Thus, ranges of 50 mV, 500 mV and 5 V can be selected. In all three cases the voltage divider guarantees a voltage of

[1] Datasheet: TIDA-01012

approximately 50 mV as seen in equation 3.1.

$$
\begin{aligned}
V_{50\,\mathrm{mV}} &= \frac{R48}{R48 + R41} \cdot 50\,\mathrm{mV} \\
&= \frac{100\,\mathrm{M\Omega}}{100\,\mathrm{M\Omega} + 10\,\mathrm{M\Omega}} \cdot 50\,\mathrm{mV} \approx \underline{45.5\,\mathrm{mV}} \\
V_{500\,\mathrm{mV}} &= \frac{R48 \parallel (R43 + R49)}{R41 + (R48 \parallel (R43 + R49))} \cdot 500\,\mathrm{mV} \\
&= \frac{\frac{100\,\mathrm{M\Omega} \cdot 1.0102\,\mathrm{M\Omega}}{100\,\mathrm{M\Omega} + 1.0102\,\mathrm{M\Omega}}}{10\,\mathrm{M\Omega} + \frac{100\,\mathrm{M\Omega} \cdot 1.0102\,\mathrm{M\Omega}}{100\,\mathrm{M\Omega} + 1.0102\,\mathrm{M\Omega}}} \cdot 500\,\mathrm{mV} \approx \underline{45.5\,\mathrm{mV}} \\
V_{5\,\mathrm{V}} &= \frac{R48 \parallel (R43 + R49) \parallel R44}{R41 + (R48 \parallel (R43 + R49) \parallel R44)} \cdot 5\,\mathrm{V} \\
&= \frac{\left(\frac{1}{100\,\mathrm{M\Omega}} + \frac{1}{1.0102\,\mathrm{M\Omega}} + \frac{1}{100\,\mathrm{k\Omega}}\right)^{-1}}{10\,\mathrm{M\Omega} + \left(\frac{1}{100\,\mathrm{M\Omega}} + \frac{1}{1.0102\,\mathrm{M\Omega}} + \frac{1}{100\,\mathrm{k\Omega}}\right)^{-1}} \cdot 5\,\mathrm{V} \approx \underline{45\,\mathrm{mV}}
\end{aligned}
\tag{3.1}
$$

A voltage difference of maximum 50 mV is applied between pins 3 and 5 of the operational amplifier OPA2313ID, which serves as a voltage follower. Afterwards, the fully differential amplifier is configured as an inverting amplifier with gain of $G = -\frac{R46}{R45} = -\frac{130\,\mathrm{k\Omega}}{3.01\,\mathrm{k\Omega}} = -43.2$. Thus, an absolute voltage difference of $50\,\mathrm{mV} \cdot 43.2 \approx 2.2\,\mathrm{V}$ is available at the capacitor in front of the ADS8885. This fits the reference voltage of 2.5 V. The capacitor C45 and the two resistors R47 and R55 form a low pass filter. The values of the resistances and capacitors were adopted by Texas Instrument[2]. The $0\,\Omega$ resistors decouple the different parts of the measurement and can be unmounted for troubleshooting.

3.9 Current Measurement

The current measurement seen in figure 3.8 is similar to the voltage measurement. Compared to figure 3.7, the right hand side of the current measurement is exactly the same. Again, a voltage of approximately 2.2 V is achieved at the pins of the ADS8885. The current range can be selected either by short-circuiting the resistor R12 or not and measure the voltage drop across resistor R15 or both R12 and R15 together.

[2]Datasheet: TIDA-01012

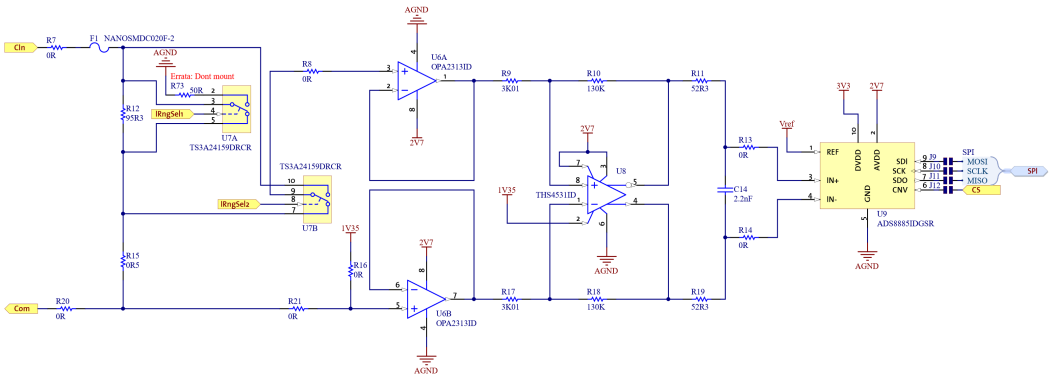


Figure 3.8: Blockdiagram of current measurement.

3.10 PCB

The PCB has a total of four layers, two of which are signal layers, a ground plane and one power plane. The ground plane is divided into an analog and digital ground and are coupled across a $0\,\Omega$ resistor. The power plane is divided into different sections to support all the ICs with their specific supply voltages.

Chapter 4

Software Description

4.1 User Manual

4.1.1 Connecting to the Demonstrator

To connect to the DEMONSTRATOR, open the WI-FI options of your device and connect to the "DEMONSTRATOR " SSID. After connecting, open `http://172.24.1.1` in a browser of your choice.

4.1.2 Powering off the device

Powering off the device can be done either with software (recommended) or hardware.

Software

To power off the device with software, press the power off button in the action bar. Confirm the dialog to shutdown the DEMONSTRATOR.
(If the power button is not shown, navigate to any settings view. After logging in, the button should be visible)

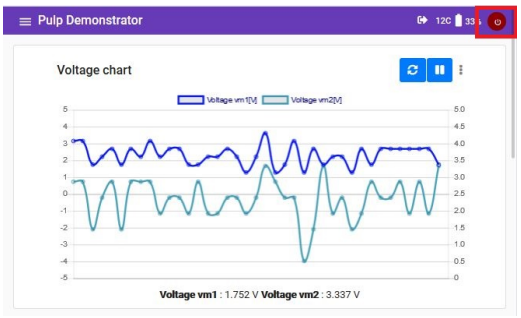


Figure 4.1: Powering off the device (1)

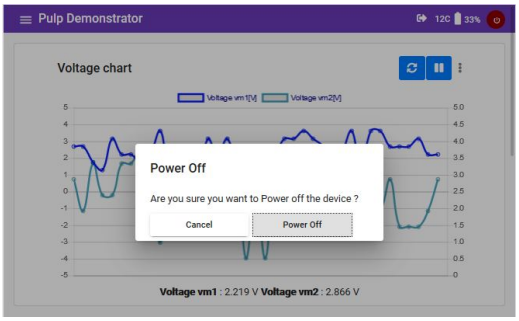


Figure 4.2: Powering off the device (2)

Hardware
To power off the device with hardware, simply press to power on/off button for approximately 20 s.
This will cut the power to the DEMONSTRATOR and it will shut down.
Only do this if you have no other option, as this could damage the device.

4.1.3 Creating or editing a chart

In order to create or edit a chart, open the sidebar and select "Settings - Chart ".

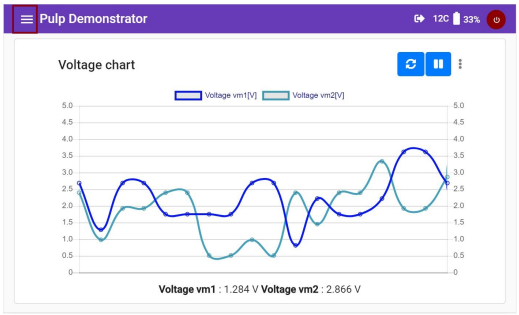


Figure 4.3: Opening the sidebar

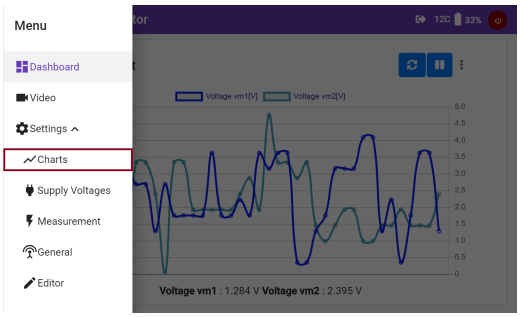


Figure 4.4: Navigating to the charts view

In the following view, select the chart you want to edit using the drop-down or create a new one by pressing the "Add Chart" button.
Changes will be applied instantly but are lost when reloading the page or restarting the server.
To store the changes globally, be sure to press the "Store changes" button.

Every chart contains 3 parts:

1. General Chart information
   - **Chart Name**: Select a unique chart name for your chart. This will also be the title of the chart, which is shown in the dashboard.
   - **Values in Graph**: Defines the max amount of data points that will be shown in this chart. Whenever a chart exceeds this amount, adding a new point to the chart, will shift it to the left by one.
2. Axes [List]
   - **Axis ID**: Unique identifier for the axis. Can be chosen freely.
   - **Position**: Position of the Axis. Currently only "left" and "right" is supported. Note that you can have more than one axis per side.

3. Dataset [List]

   - **Name**: Name of the dataset. Will be used as label in the legend of the chart.
   - **Unit**: Unit of the dataset. Will be set automatically if auto label is selected.
   - **Axis**: Axis that belongs to this dataset.
   - **Auto Label**: If set, label will be generated automatically using name and unit of the current measurement
   - **Auto Range**: If set, automatically scales the axis to fit the current resolution of the measurement(e.g. 0 - 50mV).
   - **Neg. Range**: If set, also includes negative values(e.g. -50mV - 50mV).
   - **Measurement Device**: The device which delivers the data for the dataset. If you would like to create a custom dataset (e.g. power measurement) select a random one.
   - **Color**: The color of the dataset.
   - **Apply Function:** Select this, if you want to create a custom dataset which does not simply display measured values.
     This functions gets called every time the measurement devices delivers new data.
     For more information check Section 4.1.7

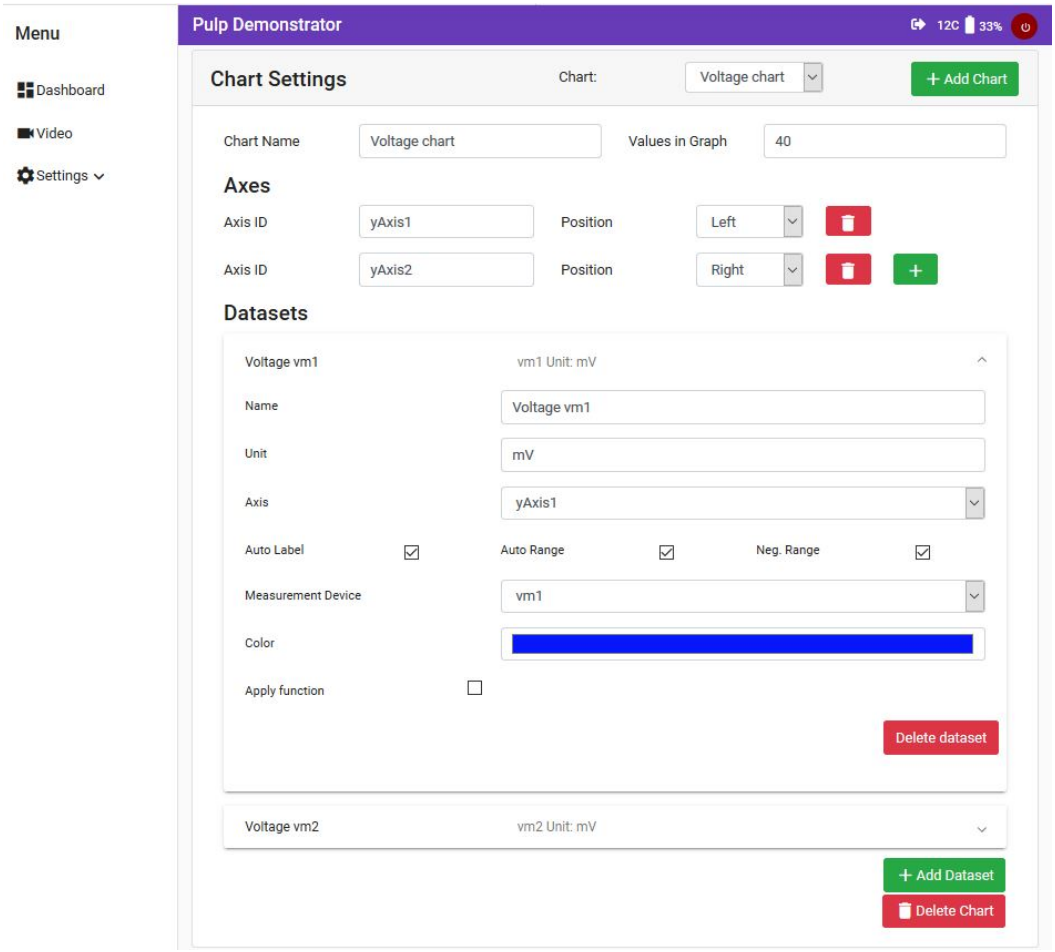


Figure 4.5: Example configuration

4.1.4 Changing the supply voltages

To change the supply voltages, navigate to "Settings - Supply Voltages".
In this view, seven different supply voltages can be configured. Use the slider or type inside the text field to change the current values.
Once you selected the desired voltages, press submit and confirm your changes.

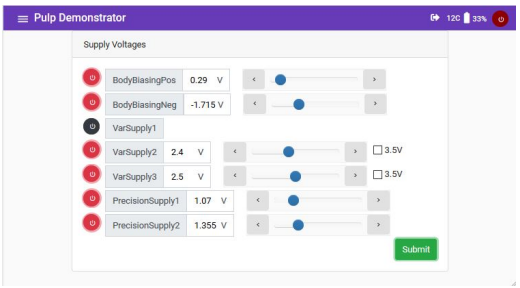


Figure 4.6: Changing the supply voltages

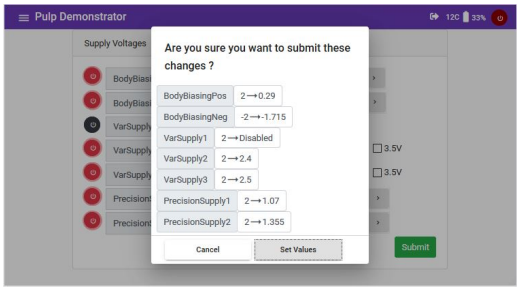


Figure 4.7: Confirming the changes

4.1.5 Changing the resolution

To change the resolution of a measurement, navigate to "Settings - Measurement".
In this view, you can adjust the resolution range of every measurement device.
If "Auto" is selected, the device tries to adjust the resolution in realtime whenever a measurement exceeds the current range.

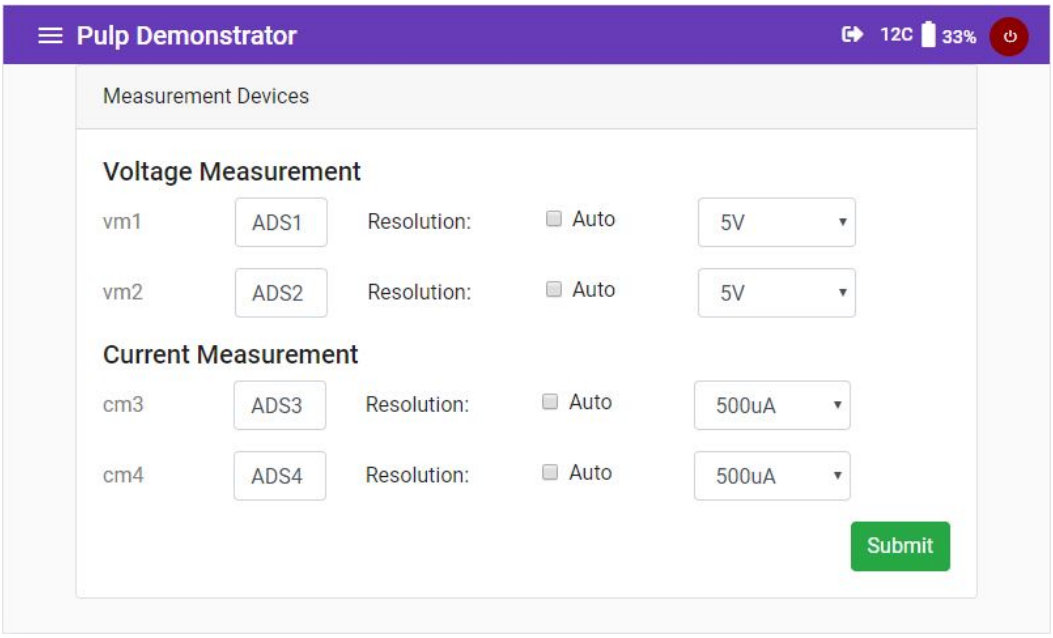


Figure 4.8: Changing the resolution of a measurement

4.1.6 Exporting a chart

Charts can be exported either as .CSV or .PNG files.
In order to export a chart, press the pause button to stop it from updating.
By pressing the three dots next to the pause button a menu providing different export actions can be opened
The exported data will be stored locally on the DEMONSTRATOR.

4.1.6.1 Obtaining the exported files using SSH

To obtain the exported files using SSH, simply connect to the DEMONSTRATOR using the SSH credentials and a SSH client of your choice.
Once connected, navigate to `/home/pi/pulp-demonstrator/Server/backend/export` to see all the files that have been exported.
The files can be downloaded using scp or can be copied to a USB stick that is plugged into the DEMONSTRATOR.

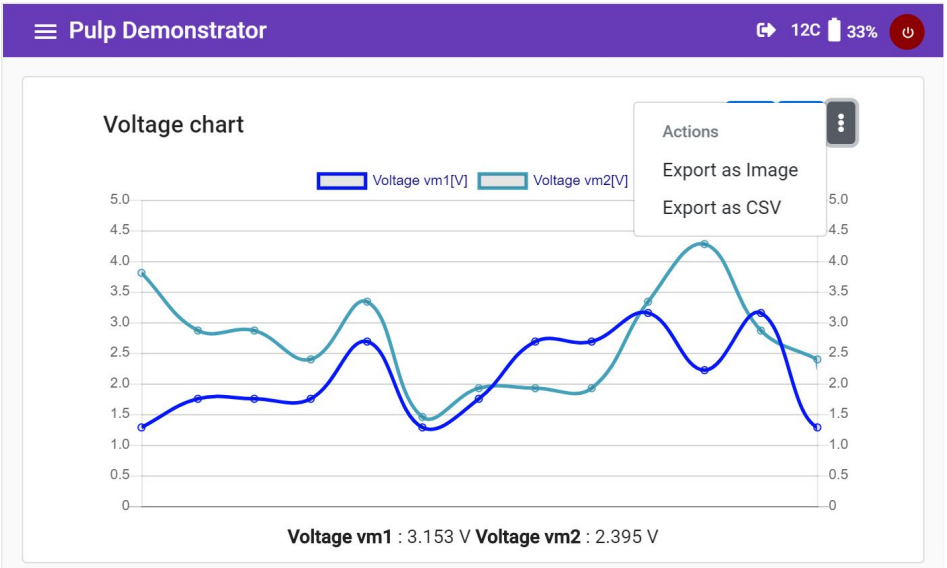


Figure 4.9: Export the data of a chart

4.1.6.2 Obtaining the exported files without remote access

To collect the exported files without remote access, plug a keyboard and a USB-Stick into the DEMONSTRATOR.
Pressing `ALT + F4` will close the GUI and opens a Linux Terminal.
Mount the USB Device using the linux `mount` [1] command.
After mounting the USB Device, navigate to
`/home/pi/pulp-demonstrator/Server/backend/export` and copy the files, you would like to export, to the USB device.

[1] `https://linuxconfig.org/howto-mount-usb-drive-in-linux`

4.1.7 Creating a custom dataset

Sometimes it is desirable to create a custom dataset that does not simply show measured data. For example, one might be interested in the power consumption of a chip, given by the multiplication of the voltage measured on the vm1 channel and the current measured on the cm3 channel.
To be able to plot datasets like these, the DEMONSTRATOR offers the possibility to apply a custom function on any dataset. This function can access any measurement and perform any operation supported by JavaScript.
To simplify development, the DEMONSTRATOR offers the function `scaleToSiUnit(DataEntry)` which takes a measurement and returns its value as an SI-Unit ([V], [A]).
All measurements are stored in the `data` object, and can be obtained using the device ID.
The following custom function plots the power consumption of the vm1 and cm3 channel in mW:

```
1 (scaleToSiUnit(data['vm1']) * scaleToSiUnit(data['cm3'])) * 1000;
```

Note that for security purposes, the function can only consist of one line. If you need to perform more complex operation, create a public function as specified in the developer manual section 4.2.

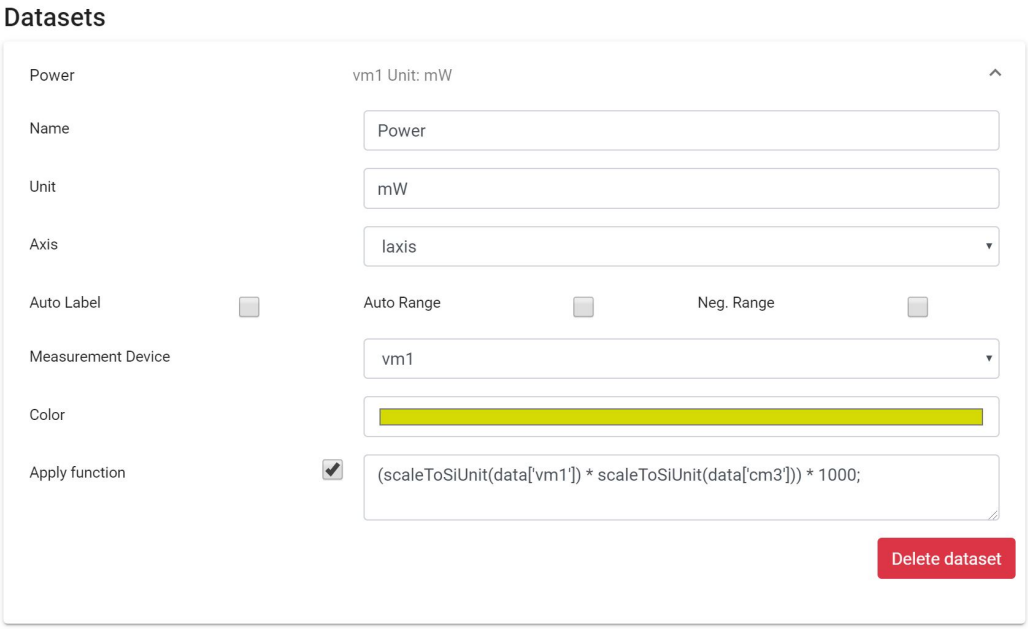


Figure 4.10: Example of a custom dataset that calculates the power consumption

4.1.8 Using the Demonstrator as logger

To use the DEMONSTRATOR as a logger, connect to the DEMONSTRATOR using SSH or by connecting a keyboard to it.
Open the Terminal and close the server task by typing:

```
# get process id of server task
ps -A | grep python
sudo kill <processId>
```

Once the server stopped, navigate to the skripts folder `pulp-demonstrator/Server/skripts/` and execute the file `perf_logger.py`:

```
cd /home/pi/pulp-demonstrator/Server/skripts
sudo python3 -d <deviceIdList> -o <outFile.csv> -t <timeDelay> -n <
    numberOfMeasurements>
```

After starting, the script will ask which resolution should be set for each measurement device specified in the deviceIdList.
As soon as the last range was specified the logger starts measuring and logging `n` values to the outputfile. The output file can then be copied on to a USB stick. The logger also creates a `<outFile.csv.info>` file which contains information about the measurement.
Examples

```
# Log the output of the vm1 and cm3 channel, waiting 0.1s between
    measurements, collecting 100 measurements.
sudo python3 -d "vm1,cm3" -o vm1_cm3.csv -t 0.1 -n 100
# Log all channels to output file. Log as fast as possible and measure
    until ctrl+c is pressed
sudo python3 -d "vm1,vm2,cm3,cm4" -o all.csv
```

4.1.9 Changing the pin-code

To change the pin-code, connect to the DEMONSTRATOR using either SSH or a keyboard.
Once connected, navigate to the file `/home/pi/pulp-demonstrator/Server/backend/flask_app` and edit the `config.py` file.
Change the line `PIN_CODE = "pincode"` to any pin-code you would like to use.
To apply the changes, restart the DEMONSTRATOR by typing: `sudo reboot now`

4.2 Developer Manual

4.2.1 Structure

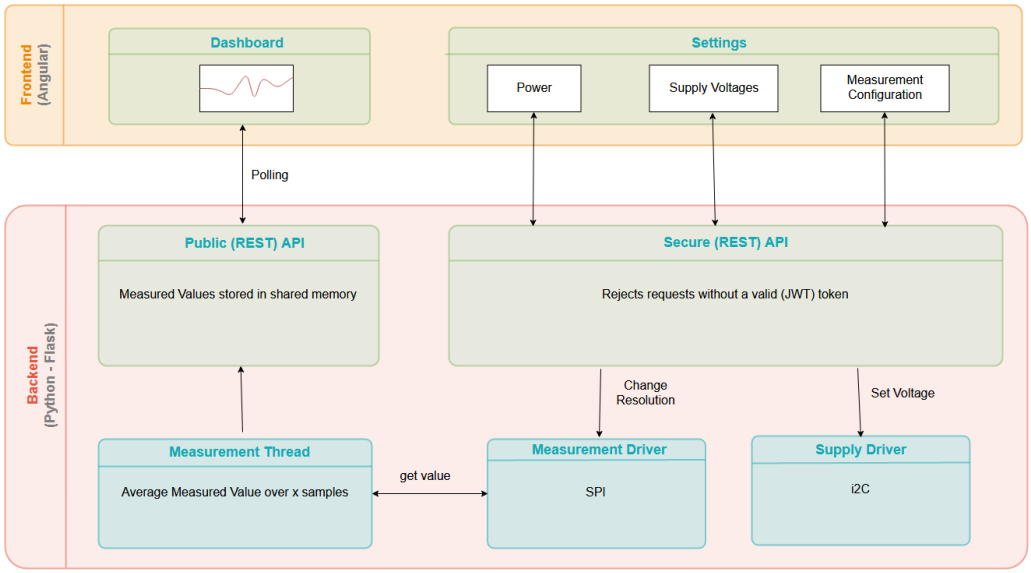


Figure 4.11: Structure of the software

4.2.1.1 Frontend

The frontend is written in Typescript[2] and uses the Angular[3] framework. Typescript is a programming language developed by Microsoft which adds static typing to the Javascript language. A Typescript file is compiled to JavaScript before it is loaded by the browser, thus it is possible to use JavaScript syntax in a Typescript file and any developer that is familiar with JavaScript will be able to make changes to the frontend.
The main part of the frontend is the Dashboard which shows different voltage and current measurements in various graphs. If needed, custom transform functions can be applied to each measurement before the value is displayed.
In order to retrieve newly measured values, the frontend polls the backend in a predefined time interval.
With this setup it is also possible to connect any device to the DEMONSTRATOR, since only a browser that is capable of running JavaScript is needed.
The second part of the frontend consists of different configuration views. These can be used to either control the seven different voltage supplies built into the DEMONSTRATOR, change the resolution of the measurement devices, create new charts or to simply turn off the DEMONSTRATOR.
These functions call "secure" routes of the API, which can only be called if the

[2] `https://www.typescriptlang.org`
[3] `https://angular.io/`

correct bearer token is submitted as part of the request.
To obtain these tokens, the `api/login` route needs to be executed with the correct pin code. These restrictions make sure that nobody can make changes to the DEMONSTRATOR without knowing the pin code and thus the DEMONSTRATOR can be used in public areas without permanent supervision.
In the following section we list all relevant files in the "front" folder and explain what they are needed for.

4.2.1.2 Frontend - Files

- **angular.json:** Contains configuration information for Angular.
- **dist:** Contains frontend builds that can be deployed.
- **documentation:** Contains the software documentation in HTML format.
- **e2e:** Contains end-to-end tests.
- **node_modules:** Contains all dependencies that are imported.
- **package.json:** Contains all dependencies that are used by the application.
- **src**
  - **favicon.ico:** Icon that is displayed in the browser tab
  - **styles.css:** General style files. Applied to the whole application.
  - **app**
    - **chart-form:** Component to edit/create charts.
    - **editor:** Component to edit the whole application configuration.
    - **general-settings:** Component to change general settings (brightness etc.).
    - **login:** Login component to enter pincode.
    - **m-dashboard:** Dashboard component containing all charts.
    - **measurement-config:** Component to configure the measurement devices
    - **navigation:** Component for sidebar and top actionbar
    - **supply-voltages:** Component to edit the supply voltages
    - **videostream:** Component to show a html videostream from backend
    - **app-routing.module.ts:** Routing module. Contains all routes ($\simeq$ URLs) that are accessible in this application.
    - **app.component.[css/html/ts]:** Root of the application. All views are shown in this component.
    - **app.module.ts:** Root module. Manages all imports, bootstraps the application.
    - **_guards:** Contains a service that checks if a user is allowed to view a given resource.
    - **_helpers**
      - **error.interceptor.ts:** Intercepts all requests. If a request returns 501 (Unauthorized) this interceptor displays the login component.

        - **jwt.interceptor.ts:** Intercepts all requests. Adds the bearer token to the request header to authenticate user.
    - **_models:** Contains models for different datatypes.
    - **_services**
        - **app.config.ts:** Loads the conf.json file from the backend and stores it for use in the GUI.
        - **authentication.service.ts:** Handles the login process.
        - **MeasurementUpdateService.ts:** Service that polls in a given time interval for new measurement data.
- **assets**
    - **hosts.json:** Contains all IP Adresses where the application should look for a backend server.
    - **conf.json:** Contains the fallback configuration of the application. Only gets loaded if no backend is found.

For further information about the source code, the documentation of the application can either be found locally (`pulp-demonstrator\Server\front\documentation\index.html`) or online at renezurbruegg.github.io/demonstrator/.

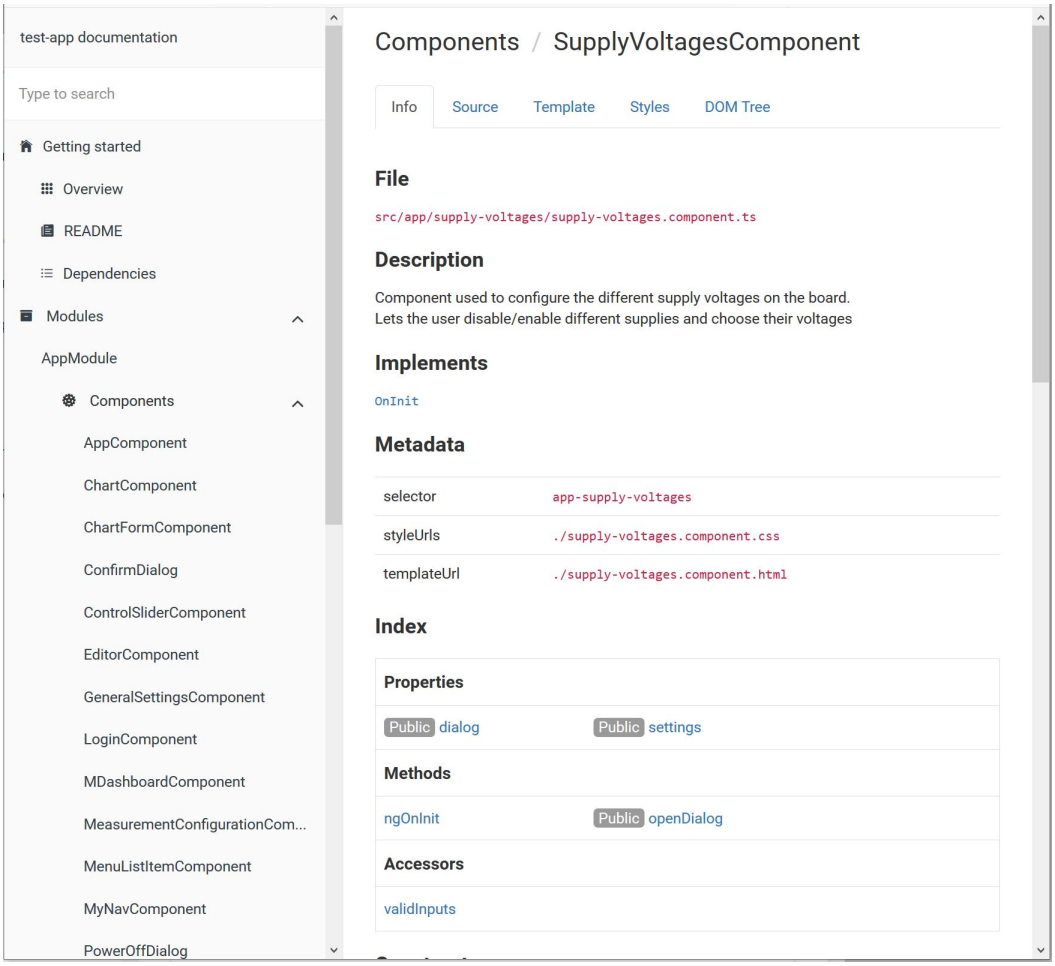


Figure 4.12: Documentation of the Angular code

4.2.1.3 Backend

The backend is written entirely in Python. It uses the Flask[4] package to create a local REST server and Flask-Security[5] to take care of the authentication process. The backend itself consists of a measurement thread that is running on a given interval that is configurable in the python code. With each cycle, the thread collects N-samples from all measurement devices configured in the devices.py file. After collecting the samples, the thread averages its values and stores it in the global `voltageValues` object. Whenever the route `/api/voltageUpdate` is requested, the server responds with the current `voltageValues` object.
Further, the backend provides some secure API routes, that are only executed if the request contains a valid JWT token. This token can only be obtained from the backend by calling the `/api/login` route with the right credentials.

4.2.1.4 Backend - Files

- **export:** Contains all exported graphs.
- **requirements.txt:** Contains all python dependencies.
- **run.py:** Starter script for the backend server.
- **logs:** Folder containing all log files.
- **flask_app**
  - **server.py:** Main File. Contains all API routes and polling thread.
  - **conf.json:** Stores the frontend configurations. (Charts, poll interval, ...)
  - **config.py:** Stores the backend configurations. (Port, PIN Code, Debug Level, ...)
  - **Devices.py:** Contains all devices that are installed on the PCB.
  - **http_codes.py:** Helper file containing most used HTTP status codes.
  - **VarSupply.py:** Interface modeling a Var-Supply.
  - **Driver**.
    - **ADS8885.py:** Driver to communicate with ADS8885 chip using SPI.
    - **LTC2631.py:** Driver to communicate with LTC2631 chip using I2C.
    - **PCAL6416.py:** Driver to communicate with PCAL6416 chip using I2C.
- **html:** Contains the .html documentation of the source code.

For further information about the source code, the documentation of the backend can either be found locally (`pulp-demonstrator\Server\backend\html`) or online.

[4] `http://flask.pocoo.org/`
[5] `https://flask-security.readthedocs.io/`

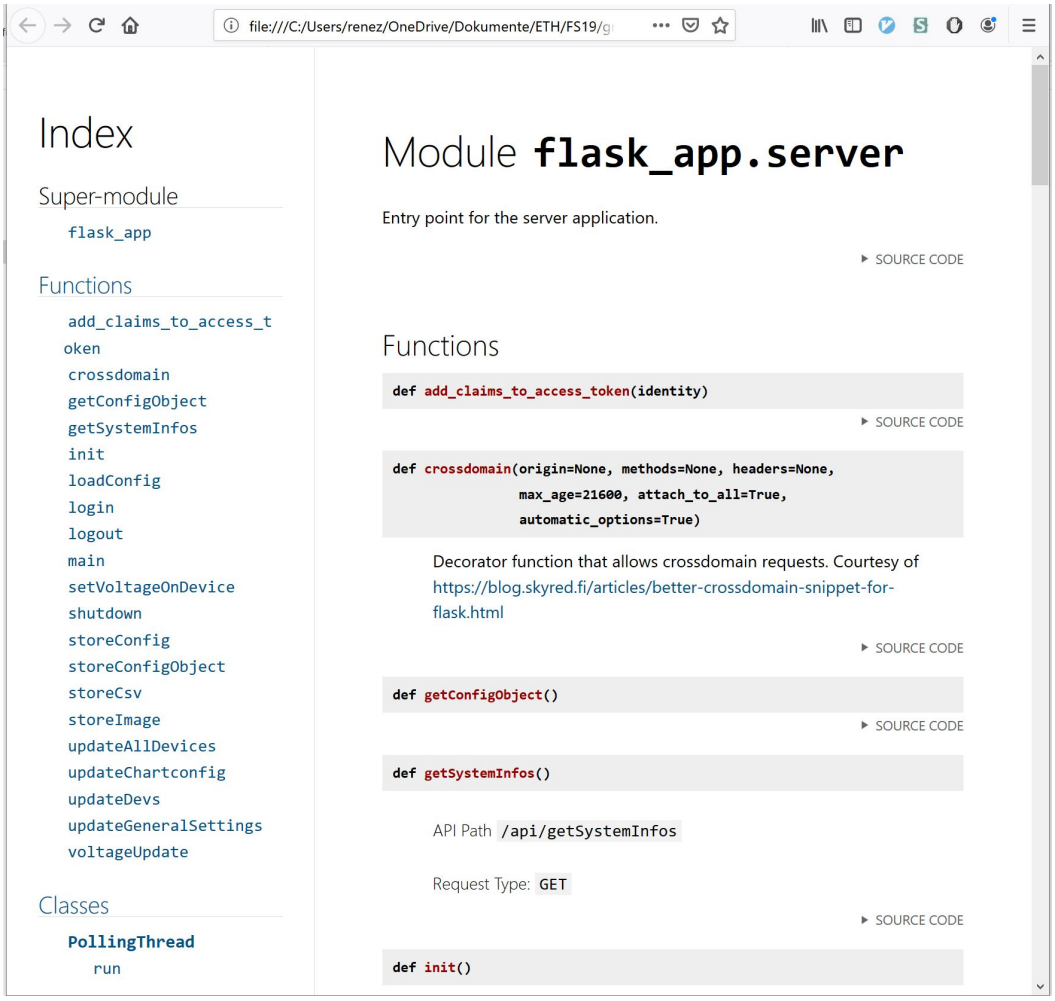


Figure 4.13: Documentation of the backend python code

4.2.2 Setting up a new Raspberry Pi

Download Raspbian Stretch Lite[6] and flash it to the SD card.
Boot the Raspberry Pi and connect it to a network via wireless or an ethernet cable. It is also recommended to enable a SSH connection using the `raspi-config` command.
Now clone the Demonstrator repository[7] to the home folder by typing
`git clone https://gitlab.ethz.ch/zrene/pulp-demonstrator.git`

```
cd ~
git clone https://gitlab.ethz.ch/zrene/pulp-demonstrator.git
```

Move to the Server folder and execute the install.sh script as root.

```
cd ~/pulp-demonstrator/Server
sudo ./install.sh
```

This will set up all necessary software and configures the device.
Sometimes the script has to be run twice to finish successfully.
After running the installation script, change the password of the Raspberry Pi.
Deploy the frontend:

```
cd ~/pulp-demonstrator/Server/skripts
sudo ./deploy.sh
```

If you plan on connecting the device to a WIFI, connect now and check what IP address your device has.
To be able to connect any device to the GUI of the Demonstrator, make sure to add the IP adress to the hosts file located at `/var/www/html/assets/hosts.json` file.

4.2.3 Enabling/Disabling the Hotspot

To enable or disable the hotspot, run the `disable_hotspot.sh` or the `enable_hotspot.sh` as root located in the `server/skripts` folder.

[6] `https://www.raspberrypi.org/downloads/raspbian/`
[7] `https://gitlab.ethz.ch/zrene/pulp-demonstrator.git`

4.2.4 Setting up the frontend to develop locally

To make changes to the frontend, it is recommended to clone the repository locally on your PC, since compiling it on the Raspberry Pi can take quite a while.
Install the Node Package Manager (NPM)[8] to be able to install the needed frontend dependencies.
Now install the ANGULAR CLI as global dependency by executing:

```
npm i -g @angular/cli
```

Move to the frontend folder (`~/pulp-demonstrator/Server/front`) and install all dependiencies that are needed to compile the GUI by typing:

```
npm install
```

The command also downloads the icon fonts and can take a while to complete.
After the command is finished, execute

```
ng server --open
```

to start the development server locally and open the GUI in your browser.
You can now start programming. Changes made to any file stored in the src folder will be compiled and shown as long as `ng serve` is running.

4.2.5 Setting up the backend to develop locally

To develop the backend, python3 and pip needs to be installed on your system.
To install the required dependencies execute the following:

```
cd ./pulp-demonstrator/Server/backend
sudo pip3 install -r requirements.txt
```

If you plan on developing on the Raspberry Pi make sure that you have run the install.sh script located in the script folder.

After setting up the dependencies, the backend server can be started by typing:

```
sudo python3 run.py
```

This will start a local (flask) server on port 8080. Verify that the server is really running by typing:

```
curl localhost:8080/api/loadconfig
```

or just opening the page in any browser.
If a JSON response is displayed, the backend server is up and running.

[8] `https://www.npmjs.com/`

4.2.6 Adding a new view to the GUI

To create a new view in the GUI, navigate to the "front" folder and create a new component using the ANGULAR CLI:

```
cd ~/pulp-demonstrator/Server/front
ng generate c <componentName>
```

This will create 4 new files inside a `src/<componentName>` folder:

- `<componentName>.component.html`
  HTML file that contains the structure and GUI elements.
- `<componentName>.component.css`
  CSS file that can be used to style the component. CSS style rules stored in this file will only be applied on the corresponding component.
- `<componentName>.component.ts`
  Main part of the component. All Typescript (or Javascript) code that belongs to this component is stored here.
- `<componentName>.component.spec.ts`
  File that stores end-to-end tests for the component

It will also create an entry in the `src/app/app.module.ts` file which contains all components.

To display the component as an item in the navigation menu, the new component has to be registered.
To do this we need to add a route to the GUI, so we can access the component using a URL.
Edit the `app-routing.module.ts` file located at `front/src/app`.
Import your component by appending the following line to the import statements:

```
import { <componentName> } from './<componentName>/<componentName>.
    component';
```

And register the component as a route by appending the following to the routes array:

```
{ path: '<componentName>', component:<componentName> }
```

Note that the path can be chosen freely.
If you want to restrict the view so only users that are logged in can see it, change the route like follows:

```
{ path: '<componentName>', component:<componentName>, canActivate: [
    AuthGuard]}
```

Now we can access the component using a URL. But we also need to add a link in the navigation menu.
To do this, navigate to the navigation component stored at `src/app/my-nav` and open the `my-nav.component.ts` file.
In this file add your component to the `NavItem` array, by appending the following:

```
{
      displayName: '<Name in Menu>',
      iconName: '<Name of a Material Icon>',
      route: '<route to your component>',
      loginRequired: <true|false>,
}
```

You should now see your component in the navigation menu

4.2.7 Deploying a new version of the GUI

To deploy a new version of the GUI the Angular source code has to be compiled and than deployed to the Raspberry Pi.
Run the following code inside the `front` folder:

```
ng build --prod
```

Angular will now build the newest version of the GUI and export it into `front/dist/test-app`.
To deploy this folder to the Raspberry Pi, first zip it by executing:

```
cd front/dist
zip -r build_<buildVersion>.zip test-app
```

Now commit the zip and pull it from the Raspberry Pi or just transfer it using a USB stick.
To deploy the zip run the deploy.sh script and delete the chromium cache:

```
# Deploy the file
cd ~/pulp-demonstrator/Server/skripts
sudo ./deploy.sh
# Clean up chromium cache
cd ~/.cache
rm -rf chromium
```

The deploy script takes the zip file from the `front/dist` folder, which was last modified. If you pull more than one .zip file, the script could deploy an old version. To avoid that, just rename the file you want to deploy, so it is the file last modified.

4.2.8 Changing the IP of the Demonstrator

To change the IP address of the DEMONSTRATOR, edit the file `Server/skripts/enable_hotspot.sh` and change the value of `BASE_IP` to an IP of your choice.
Then run `disable_hotspot.sh` followed by `enable_hotspot.sh`.

4.2.9 Trouble shooting

4.2.9.1 Frontend

- **Can't connect to frontend (connection refused)**
  Make sure the Angular development server is running (ng serve) and the port you are using is correct (4200).
  If it is not working on the RASPBERRY PI, make sure you got the right IP address and are on the same network.

- **Frontend stays blank**
  Make sure that the `ng serve` command isn't logging any errors.
  Open the browser console by pressing `Ctrl + Shift +i` and check if any error messages appear.
  This problem is often the result of a syntax error in one of the HTML file.

- **The frontend loads, but no data is displayed.**
  Make sure that the backend server is up and running.
  Make sure the IP-address of the RASPBERRY PI is stored in the hosts.json file located in the assets folder.

- **"Cors policy no 'access-control-allow-origin' header is present on the requested resource", appears as error in the Browser console.**
  Make sure to add the `@crossdomain(origin=*)` annotation to your backend function.

- **I deployed a new GUI but can't see any changes.**
  Make sure that your build.zip was the last file modified, you ran the ./deploy.sh script and deleted the chromium cache.

4.2.9.2 Backend

- **How can i restart the server without rebooting the Raspberry Pi?**

```
# get process id of server task
ps -A | grep python
sudo kill <processId>
sudo python3 /home/pi/pulp-demonstrator/Server/backend/run.py &
```

- **Where are the log files located?**
  The log files are stored in `pulp-demonstrator/Server/backend/logs`.

- **Where can I change the log level?**
  Open the `config.py` file located at `pulp-demonstrator/Server/backend/flask_app` and change the value of the `LOGGING_LEVEL` variable.

- **How can I show the log output on the console?**
  Open the `config.py` file located at `pulp-demonstrator/Server/backend/flask_app` and change the value of the `LOG_TO_CONSOLE` variable.

4.2.10 Calibrating the measurements

To calibrate the measurements as described in chapter 7, use the `calibrate.py` file located in the `skripts` folder.
Connect a stable power supply to the channel which you want to calibrate and start the script by typing:

```
sudo python3 calibrate.py -d "devId" -o "outputFileName.csv"
```

where `devId` is the ID of your measurement device (e.g. vm1,vm2,cm3,cm4).
The script will ask for the current resolution range and then continuously ask for the current input.
Every time an input value is entered, the skript will log the specified value and the measured value of the ADS8885 chip in the outputfile.
If you think you have enough values, exit the skript by typing `end` or pressing `ctrl+c`.
Now import the .csv file into the software of your choice to fit a linear approximation of type `a*x + b`.
Once you have obtained the value for `a` and `b` update the file `flask_app/Devices.py` and change the values of the `ResoltuionScaler` for the calibrated device.

4.2.11 Using a different PCB

To use a different PCB the following changes have to be made:

1. Create a driver for all the chips on the PCB and store them in the driver folder.

2. Update the `VarSupply.py` file located at `backend/flask_app`.

3. Update the `Devices.py` file, replace all old devices with the new ones from your PCB.

4. If needed (structure of Devices.py changed) modify the `_setVoltageOnDevice()` function located in the `server.py` file, to fit your need.

5. Update the conf.json file to register your new devices in the frontend.

4.2.12 Adding public functions to help transforming data

To create a custom dataset that acts on different measurements, additional functionality may be needed.
It is possible, to declare public functions in the frontend, which can later be used to transform data. In this section, we well show how to register a new function that can be used in the GUI.
Suppose we want to create a new custom dataset, that is always 1 if vm1 exceeds a given threshold and is 0 otherwise.
To be able to do this, we want to create a custom function `stepDetect(sourceId, threshold, measuredData)` which returns 1 if the voltage from `sourceId` exceeds the `threshold` or 0 otherwise. This function should later be used in the GUI to create a custom dataset.
To achieve this, we have to create a new **public** function in the frontend, that can be used in any Chart.
Open the file `/front/src/app/m-dashboard/chart/chart.component.ts` and create a new function by inserting the following snippet under the import statements:

```
1 export function stepDetect(sourceId:string, threshold: number,
    measuredData: Record<string, DataEntry> ) : number {
2     // get DataEntry which contains measurement that belongs to the
    source id
3     let entry : DataEntry = measuredData[sourceId];
4     // scale value to si unit
5     let scaledValue : number = scaleToSiUnit(entry);
6     // perform step detection and return 1 or 0
7     return scaledValue >= threshold;
8 }
```

Now, after compiling and deploying the application, this function can be used inside the Chart Editor.
It is also possible to create a buffer variable in the `chart.component.ts` file. This enables various functions (e.g. convolutions, AC-coupling) to be implemented.

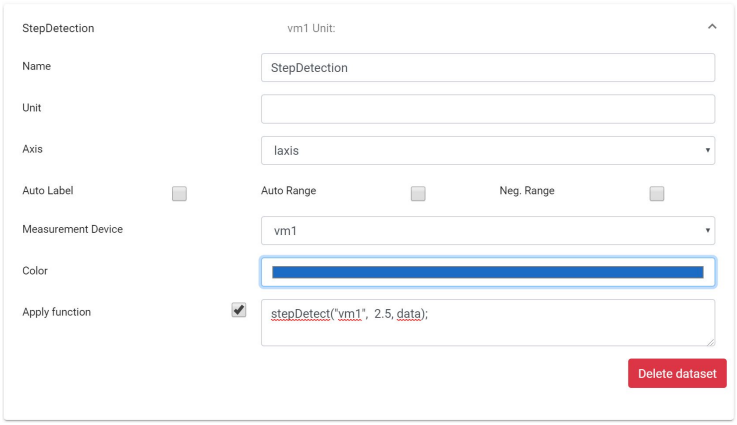


Figure 4.14: Example of the edge detection function used in the GUI

4.2.13 Change sampling rate

To change the sampling rate for the frontend, navigate to Settings-Editor and scroll down to the general properties. Change the value of the `timerInterval` to any poll interval (ms) and press the save button.

4.2.14 Frontend configuration file - conf.json

The conf.json file contains all information that is displayed in the frontend. It consists of four main parts:

1. **Charts:** Contains all information about the charts that are displayed.
2. **Deployment:** Contains all information about the backend. (Protocol used, path to api, hostname etc.)
3. **General:** Contains general information (e.g. Poll interval, custom options for all charts).
4. **Supplies:** Contains all voltage supplies that can be used on the PCB.

You can find a full documentation of the file online at renezurbruegg.github.io/demonstrator/

Chapter 5

Mechanical Setup

All hardware components are packed into a case that we designed during the group project. The housing is made of aluminum which makes it stable and reduces the chance of it breaking. Additionally the shiny aluminum gives the case a nice look.

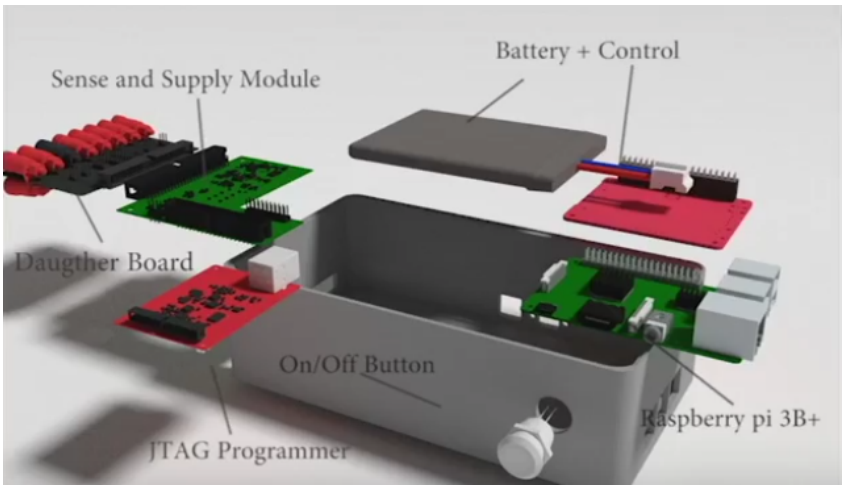


Figure 5.1: Assembly of the Demonstrator.

5.1 Mechanical Drawing of the Case

In this section you find mechanical drawings of the housing from different perspectives. On the drawings you find all the measurements (always given in millimeters) as well as the sizes of all the threads (ISO standard).

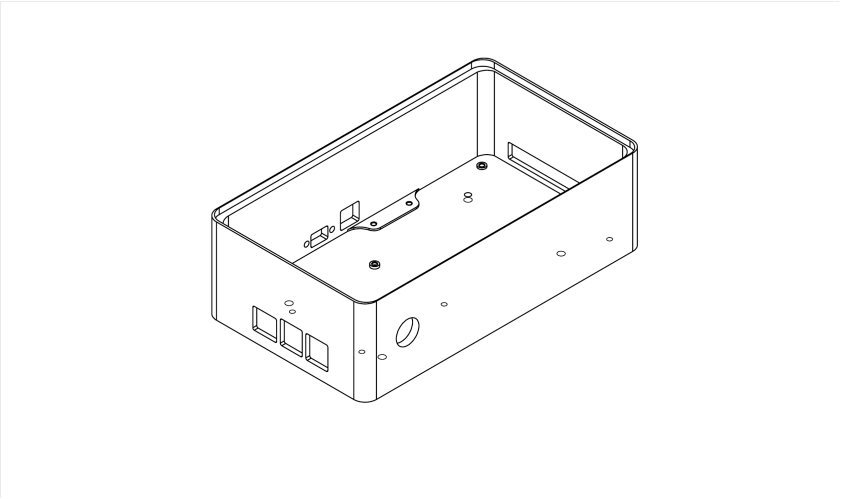


Figure 5.2: Mechanical drawing of the pulp demonstrator.

As you can see from this perspective on the top of the device the border is much thinner than on the bottom. This way the 7" touch display can be laid nicely on top of the thicker border, where as the tiny boarder surrounds the display and keeps it in the right position. Additionally, this way the monitor is protected by the small border.
By looking at the case from the top it looks like this.

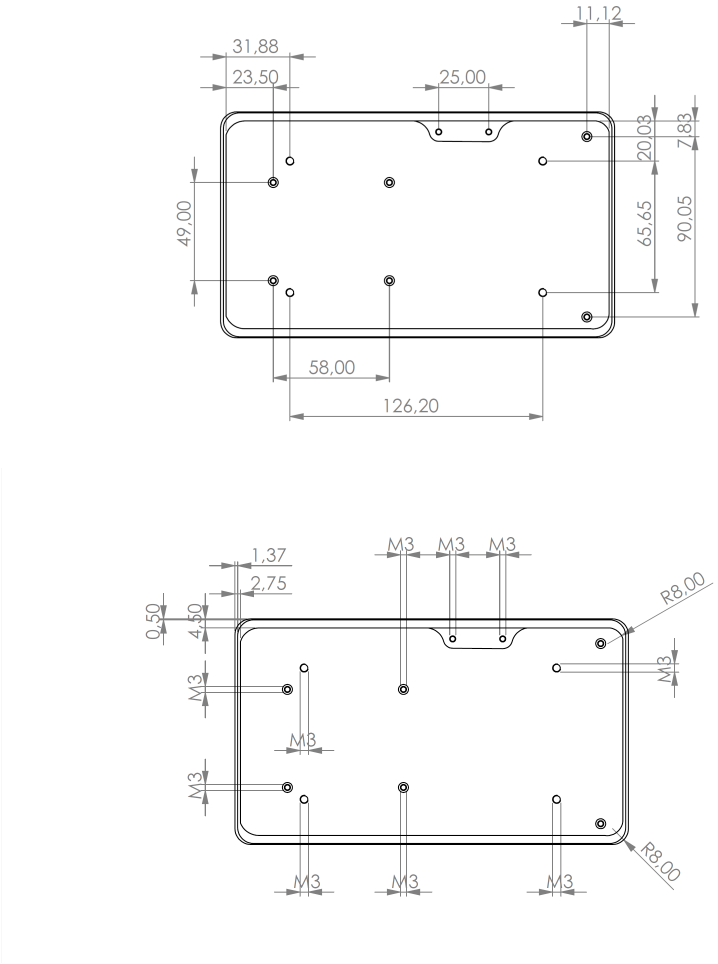


Figure 5.3: Mechanical drawing top view.

There are four threads that go through the bottom plane (see figure 5.4). They are required to fixate the touch display of the device with four screws, that go through the whole device.
Additionally there are eight M3 threads that don't drill through the whole bottom, which means that you can't see them from the outside of the housing (see figure 5.4). The reason for this is just for the beauty of the case. To make these threads long enough but still not throughout, there are risings of 1 mm height where the threads are located.
These eight threads are used to fixate the hardware components such as the Raspberry Pi, the Sense & Supply Module and the Olimex ARM-USB-OCD-H programmer component at the bottom of the Demonstrator in order to stay in place in case the device is turned upside down.

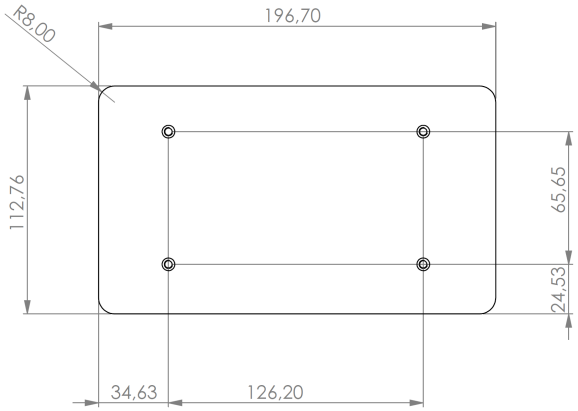


Figure 5.4: Mechanical drawing bottom view.

As already mentioned from the bottom view we only see 4 threads that fix the touch display.

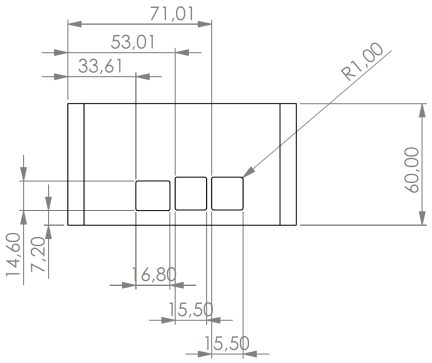


Figure 5.5: Mechanical drawing side view.

On one side of the Demonstrator you can find 3 openings. Two of them give access to the USB connections of the Raspberry Pi. The opening on the very left is needed to access the Ethernet connection of the Raspberry Pi.

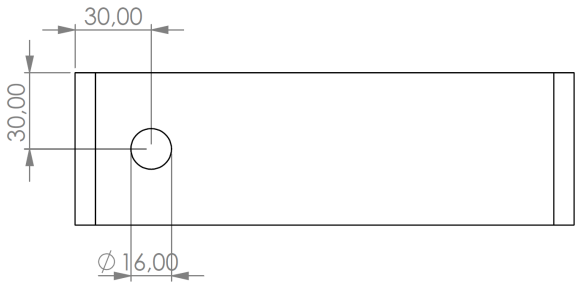


Figure 5.6: Mechanical drawing side view.

On this side of the Demonstrator we can find the button to turn on and off the device. There is a hole with a 8mm radius without any thread. The button is then attached from inside with a nut.

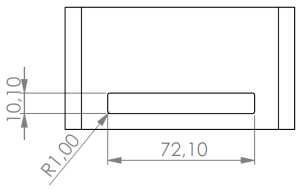


Figure 5.7: Mechanical drawing side view.

Here you can see the opening which gives us access to the pin header of the Sense & Supply Module of the Demonstrator.

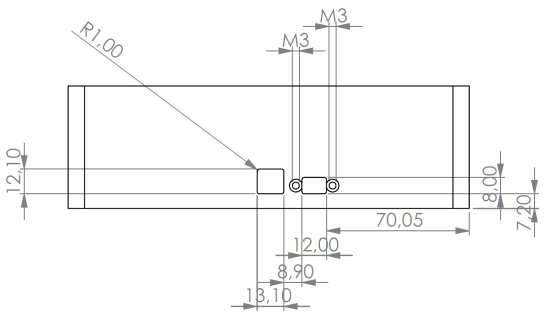


Figure 5.8: Mechanical drawing side view.

The opening on the left is needed to access the programmer. On the right hand side a micro USB port will be mounted where you can charge the DEMONSTRATOR. As the micro USB connection of the RASPBERRY PI isn't accessible directly from outside of the case, an extension cord is needed. This extension cord is then fixated with two M3 screws from the outside as you can see on the mechanical drawing.

5.2 Mounts

The 12000mAh battery is located right under the touch display in the DEMONSTRATOR. In order to fixate the battery we designed two mounts which were printed with a 3D printer. The battery is laid into the two mounts and the mounts are then screwed to the display.

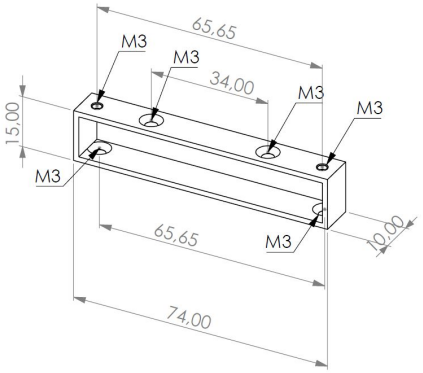


Figure 5.9: Mechanical drawing of mount A.

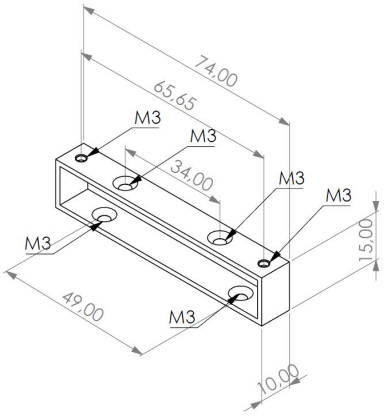


Figure 5.10: Mechanical drawing of mount B.

On the bottom of each mount there are two holes which are used to attach the mounts to the touch display.
Thanks to the other two holes that are on top of the mounts we can fixate the battery with screws.

Figure 5.11: Assembly of display and battery.

Chapter 6

Applications

At the beginning of the group project the goal was to create a device which can measure currents and voltages of a PULP chip. In the end we created a tablet, which is able to measure these values from any source.
In order to use a custom chip with the Demonstrator a Daughter Board has to be designed.
This Daughter Board simply connects each output pin of the chip to the corresponding pin on the Demonstrator.
The arrangement of the PINs and the required connector can be obtained from the following data sheet (Figure 3.2).

6.1 Daughter-Board

Our group project involved creating a naive breakout board that not only exposes the output PINs of the Demonstrator but also can be used to test the device.
For easy access, so called "banana connectors" have been installed. These connectors are widely used and therefore most people should be able to easily connect to them.
Since to Daughter Board does not expose all supplies or measurement channels, a jumper can be used to switch between different channels.
The schematics and a rendered image of the Daughter Board can be seen on the next page.

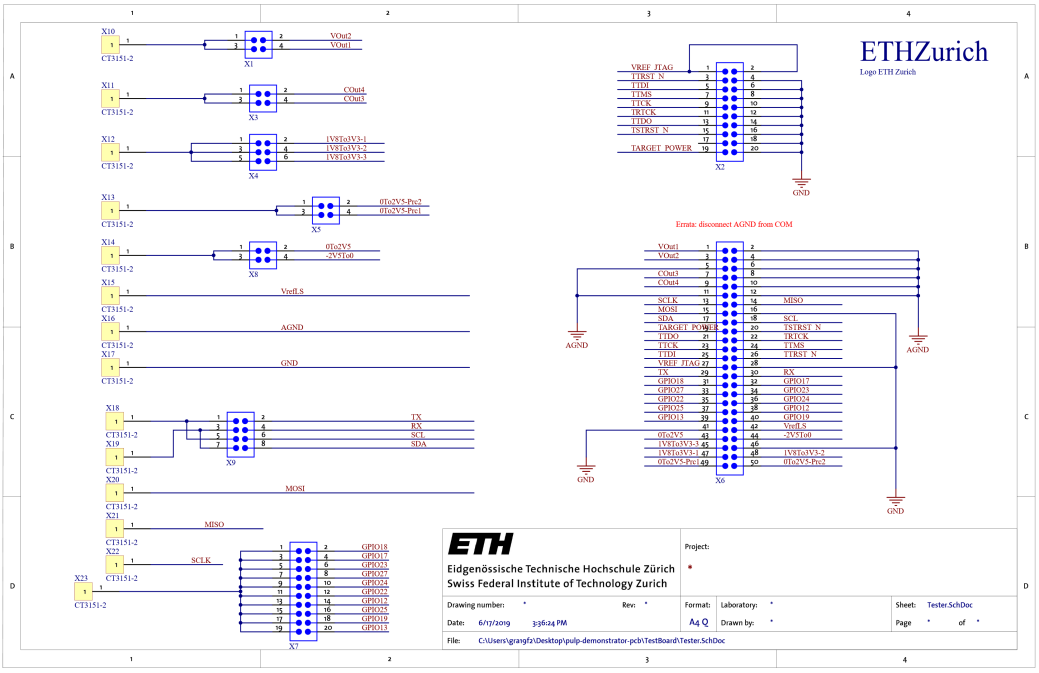


Figure 6.1: Block diagram of the Daughter Board.

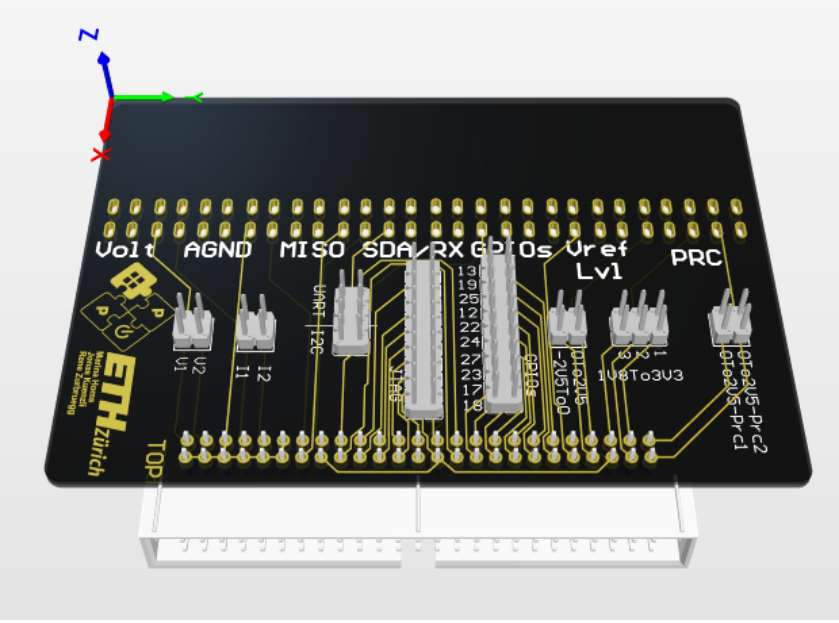


Figure 6.2: Rendered image of the Daughter Board

Chapter 7

Results

7.1 Calibration

Because the resistors, capacitors and other components we used for our Sense & Supply Module aren't ideal and have some imperfections, we had to calibrate our device, in order to get accurate measurement results. For this purpose, we connected the Demonstrator to a National Instruments NI USB-6216, with which we generated different voltages very precisely. We then measured these voltages with our Demonstrator. In the following plots the voltages generated by the National Instruments NI USB-6216 vs the values the ADS8885 chip of the Demonstrator respectively can be seen. We did the measurements for the three different measurement ranges (5 V, 500 mV and 50 mV) of both the measurement channel 1 and 2 (VM1 and VM2) as you can see in the following plots:

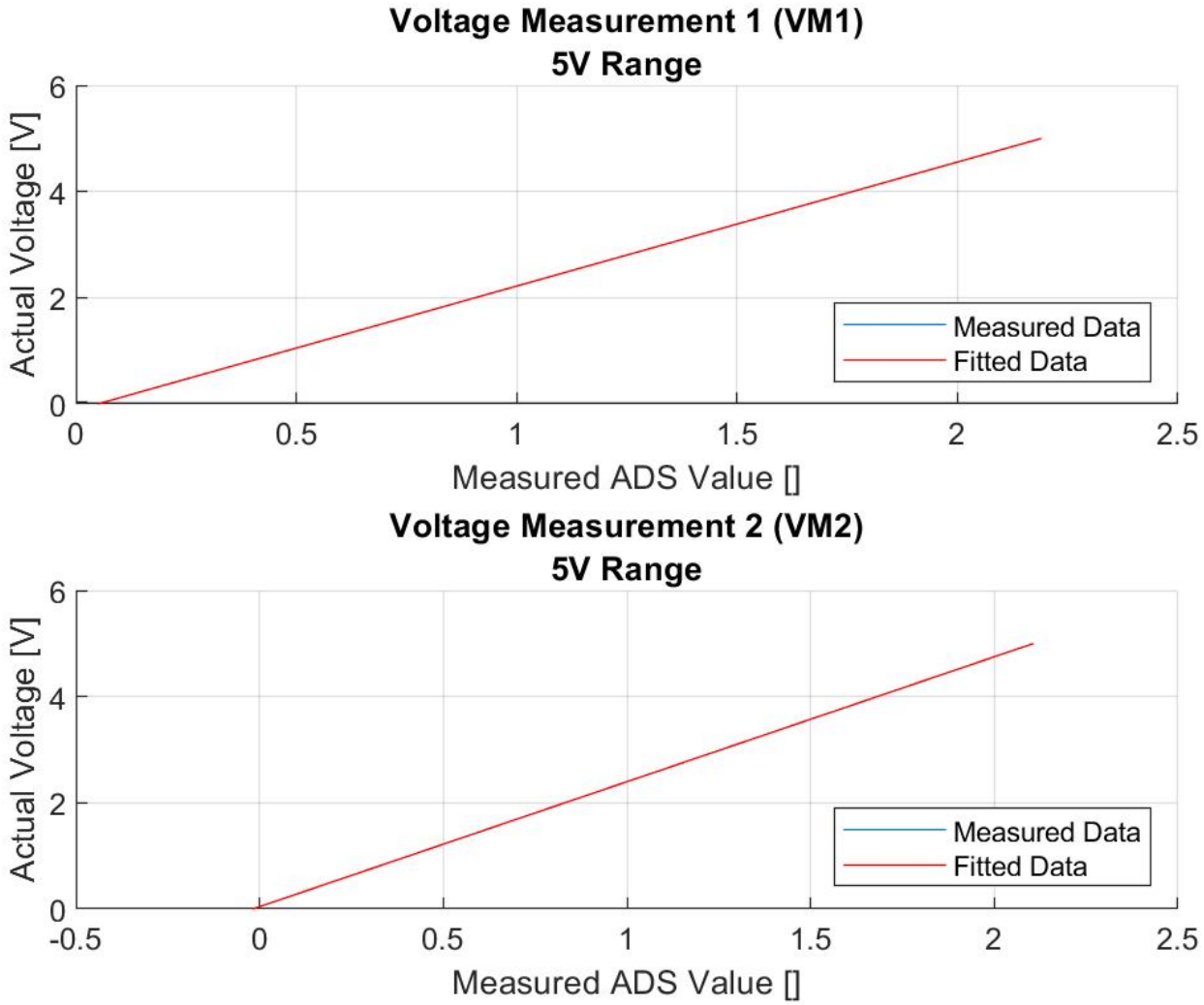


Figure 7.1: Calibration 5 V range.

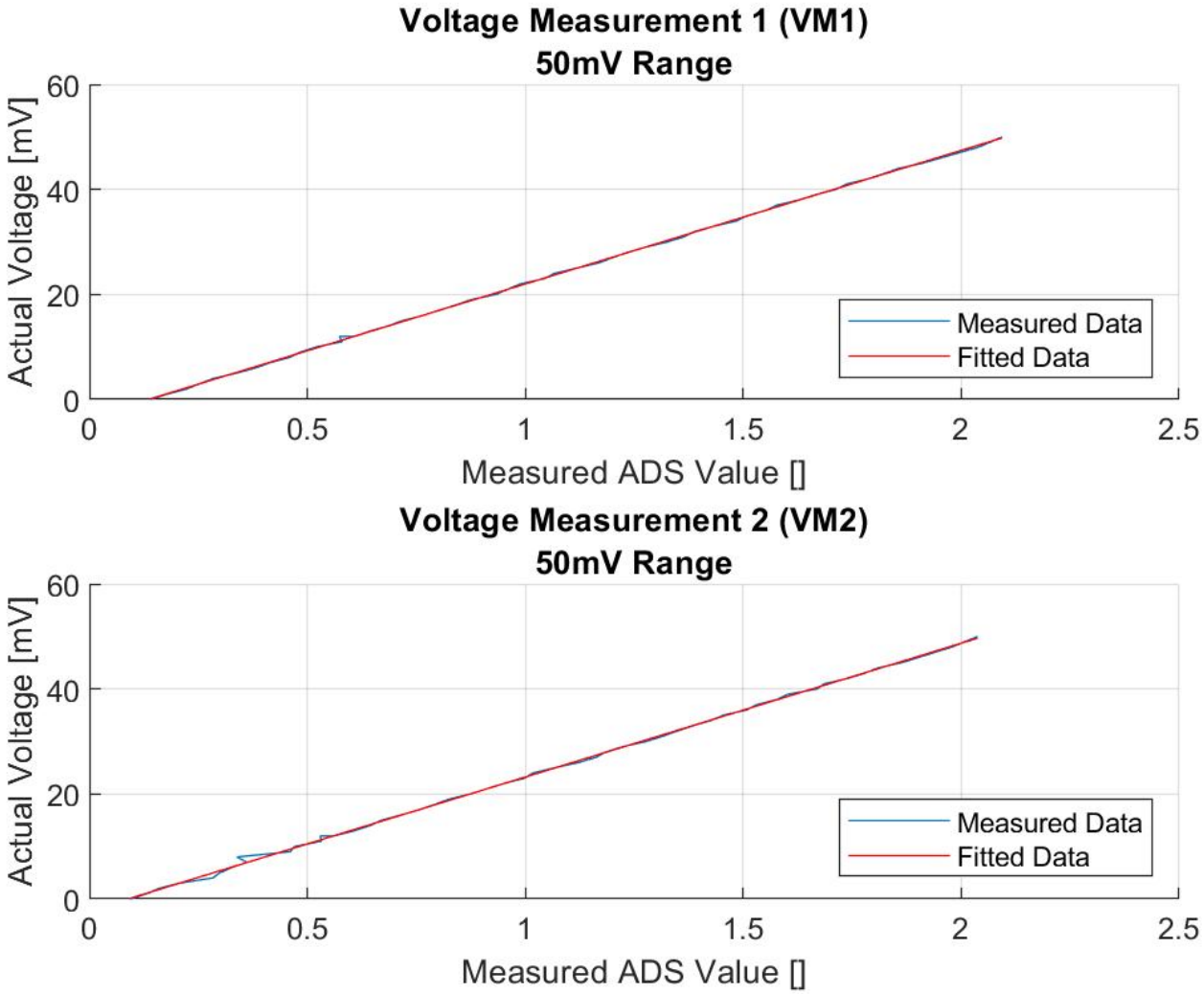


Figure 7.2: Calibration 50 mV range.

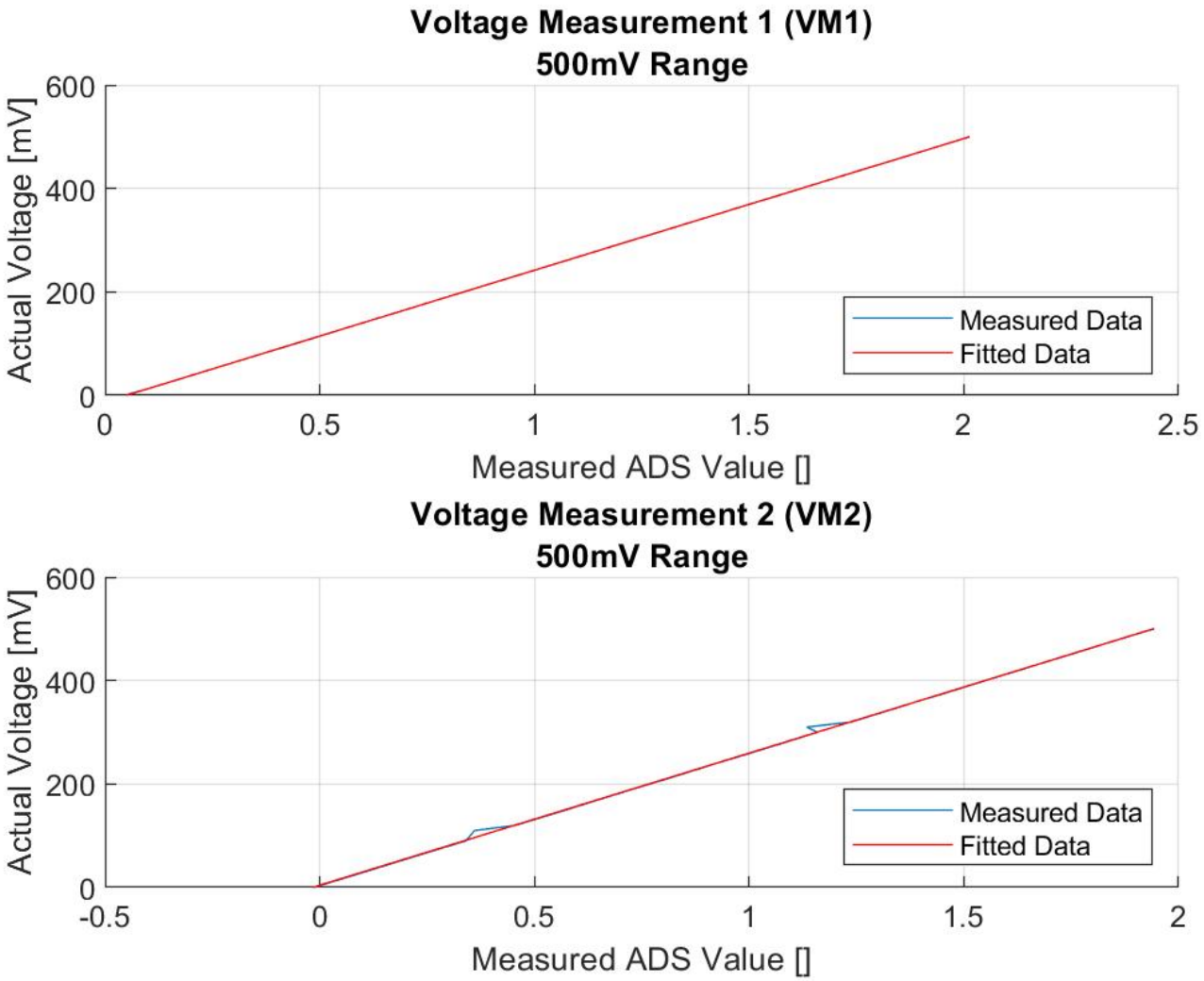


Figure 7.3: Calibration 500 mV range.

As you can see, the applied vs the measured values have a linear relation. We approximated the received data by a first order polynomial (see the blue line in the figures) to get the coefficients α and β that describe the function. With these we could adjust the measured data of the DEMONSTRATOR in the software accordingly and got precise measurement results (see chapter 7.3 and 7.2).

$$\text{applied Voltage} = \alpha \cdot \text{ADS Value} + \beta$$

Analogously, we did the same procedure for the current measurements. We generated exact known currents with the TTI PL303 and compared these with the values the ADS8885 chip of the DEMONSTRATOR measured. Then we calculated the coefficients needed for the calibration.

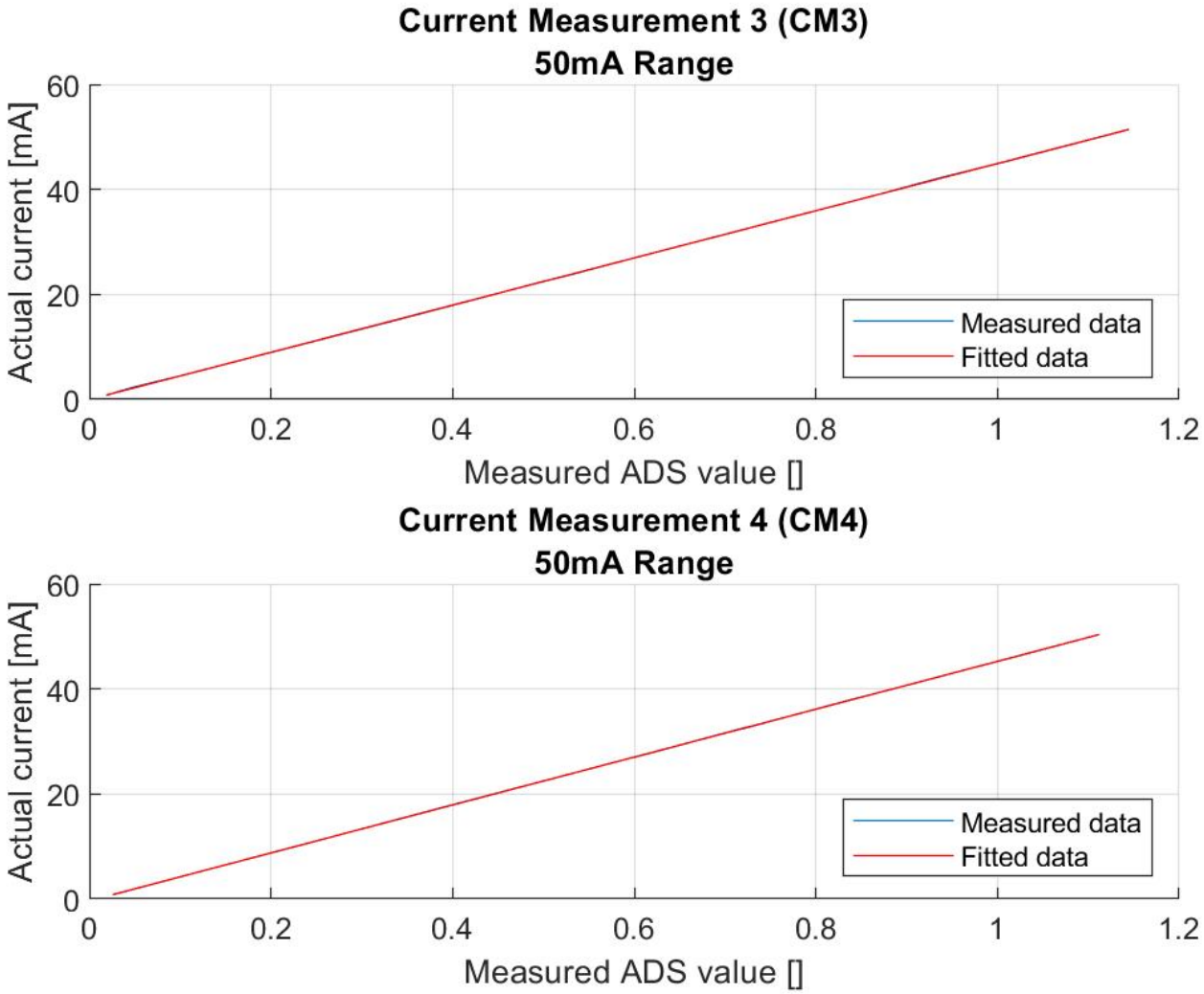


Figure 7.4: Calibration 50 mA range.

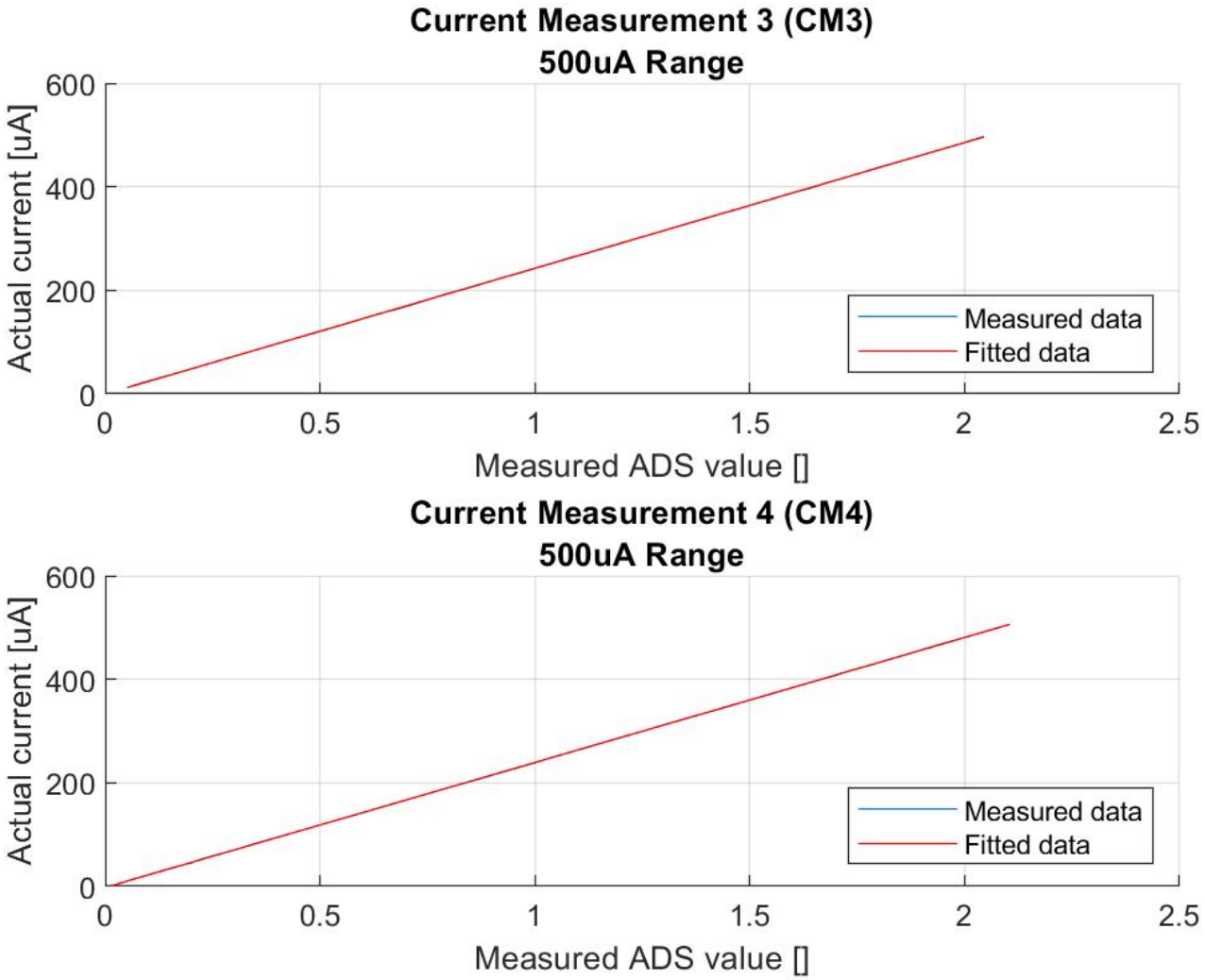


Figure 7.5: Calibration 500 µA Range

7.2 Voltage Measurement

In order to detect the accuracy of the voltage measurement, we let the Demonstrator measure a certain voltage 1000 times in a row with 0.1 s break between two measurements. We applied the voltage with the Precision Supply Keysight B2912A. With those measurements we could detect the mean value and the standard deviation with Matlab. In figure 7.6 the results in the different voltage ranges with the corresponding voltage measurement points can be seen. The blue box indicates the 25 percentiles of the data and the '+' shows the outliers[1].

[1] `https://www.mathworks.com/help/stats/boxplot.html`

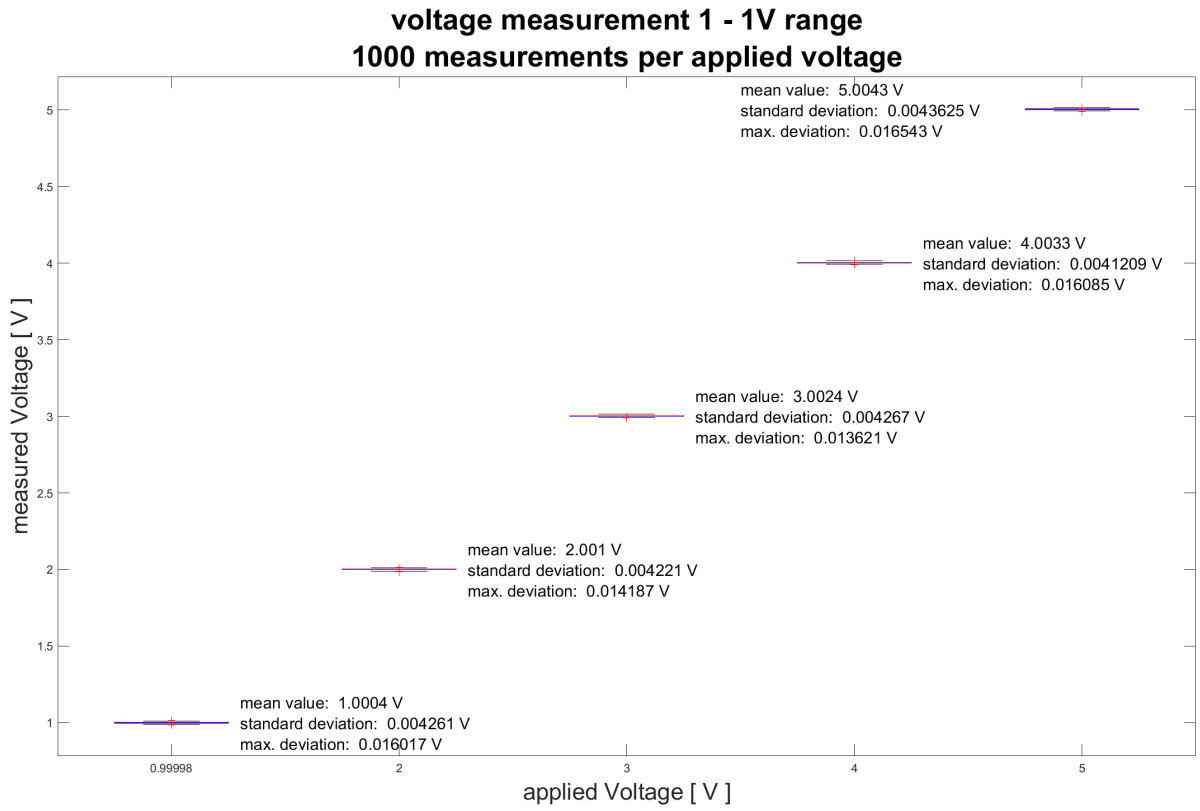


Figure 7.6: Voltage Measurement 1 - 1 V range

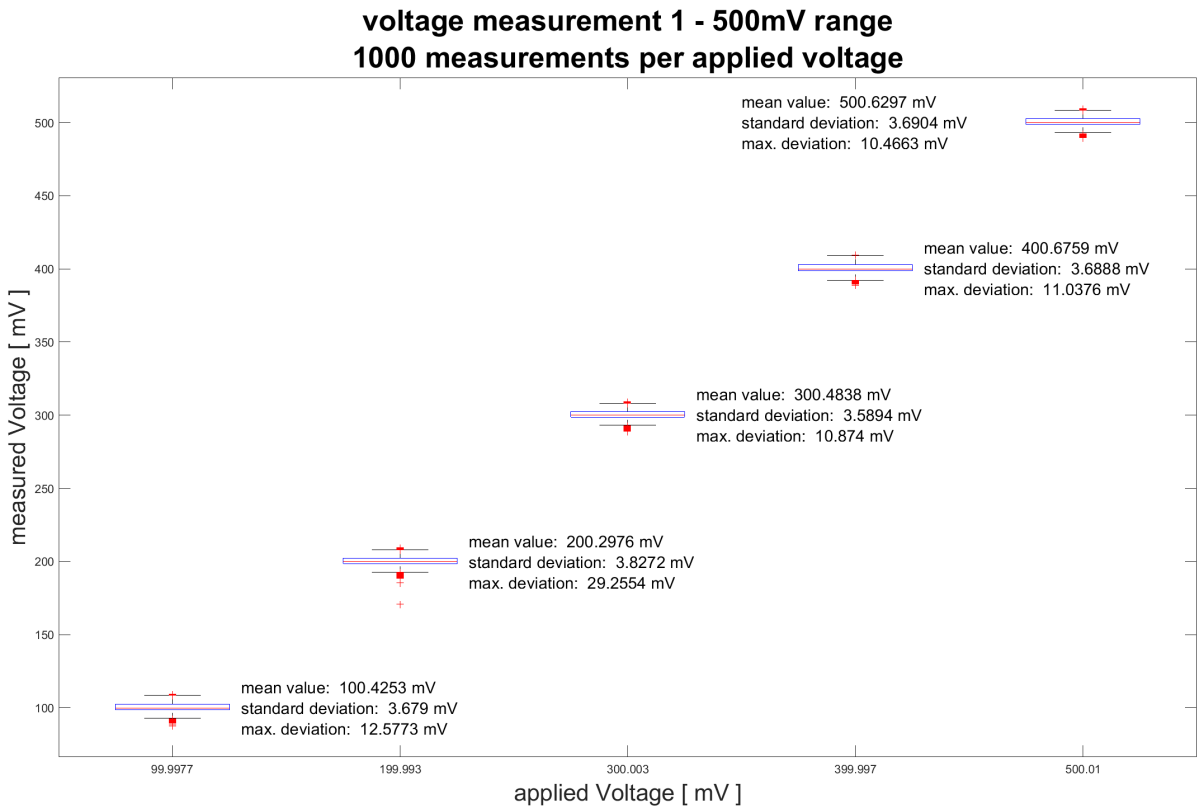


Figure 7.7: Voltage Measurement 1 - 500 mV range

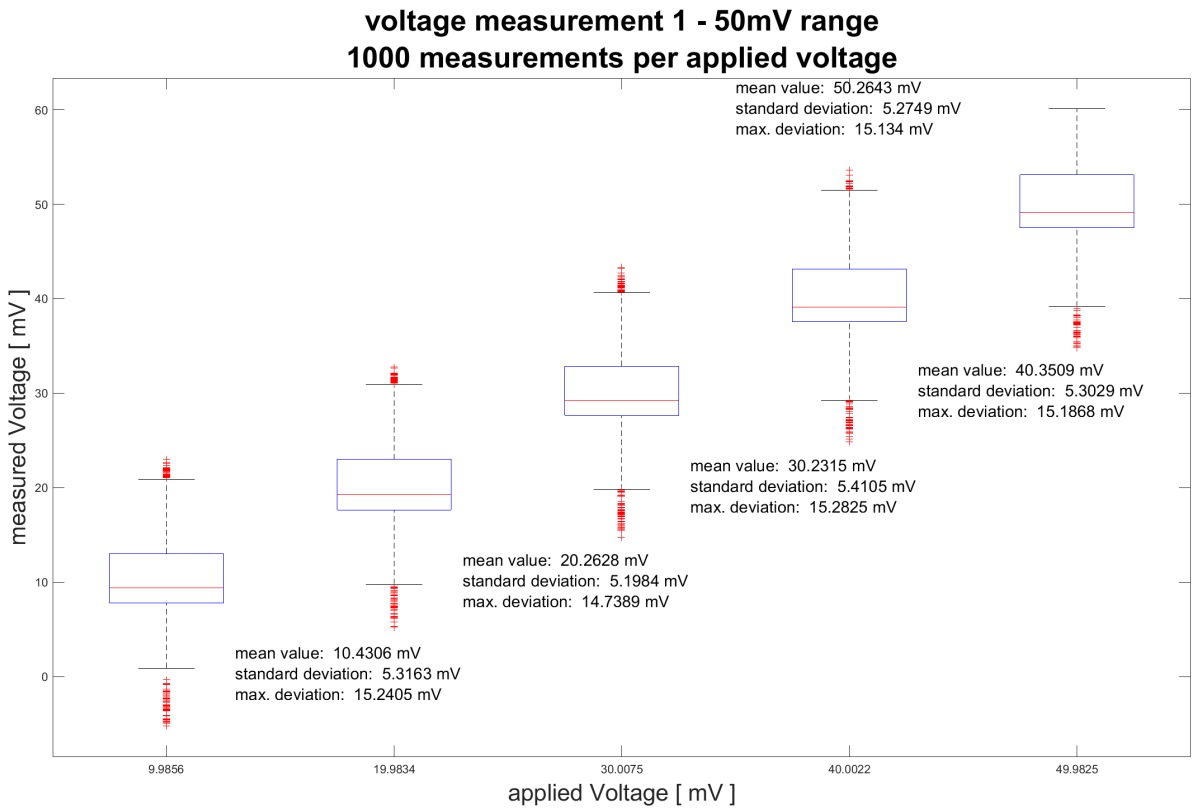


Figure 7.8: Voltage Measurement 1 - 50 mV range

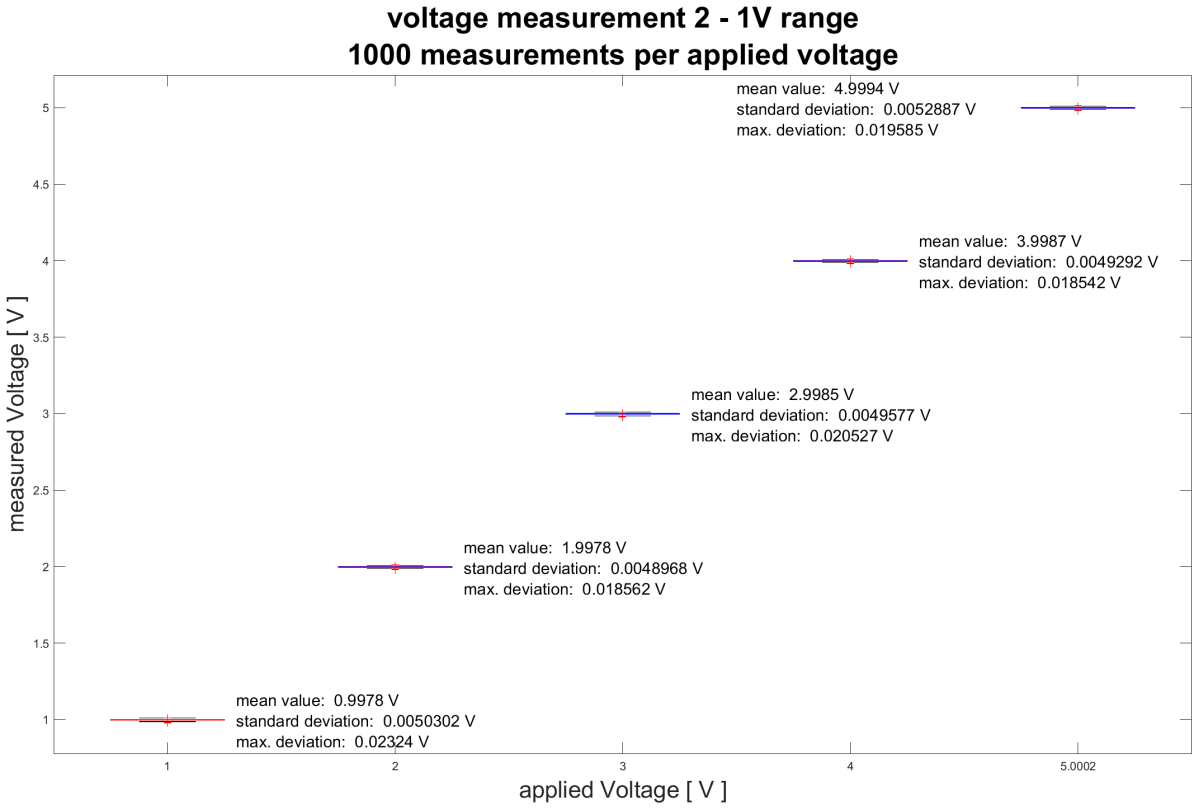


Figure 7.9: Voltage Measurement 2 - 1 V

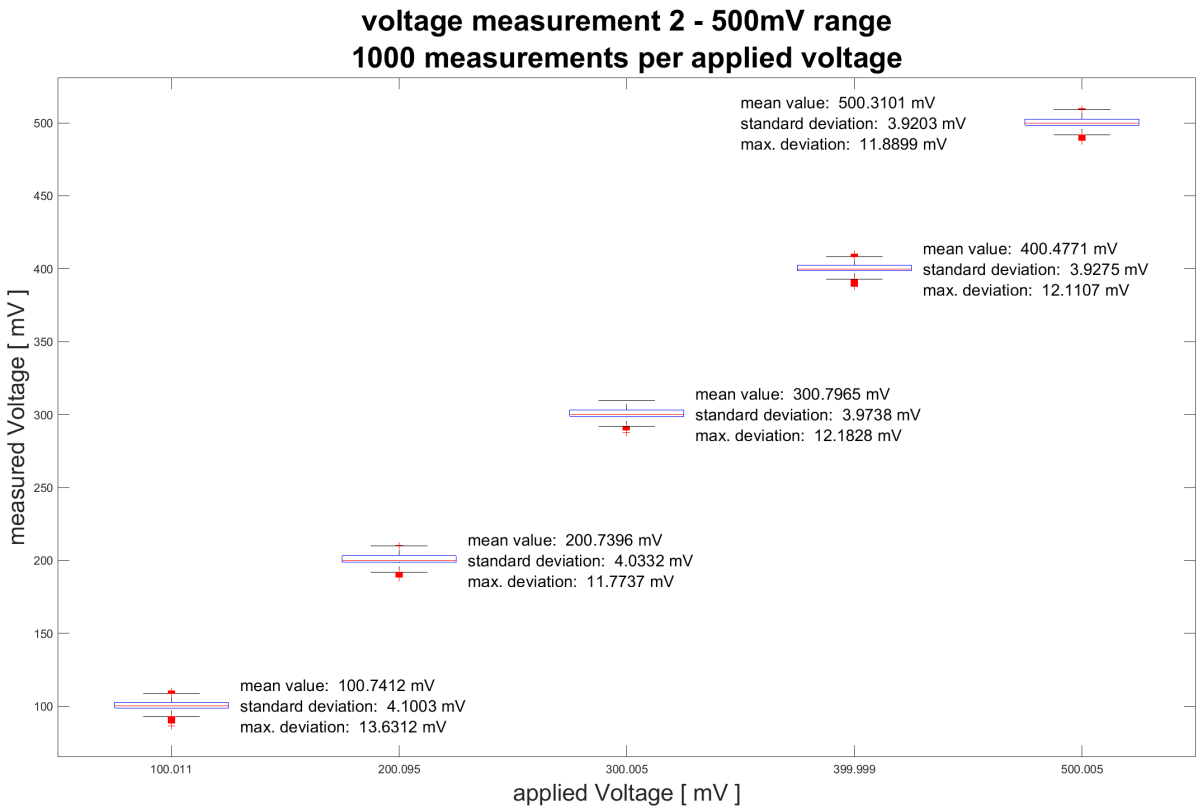


Figure 7.10: Voltage Measurement 2 - 500 mV range

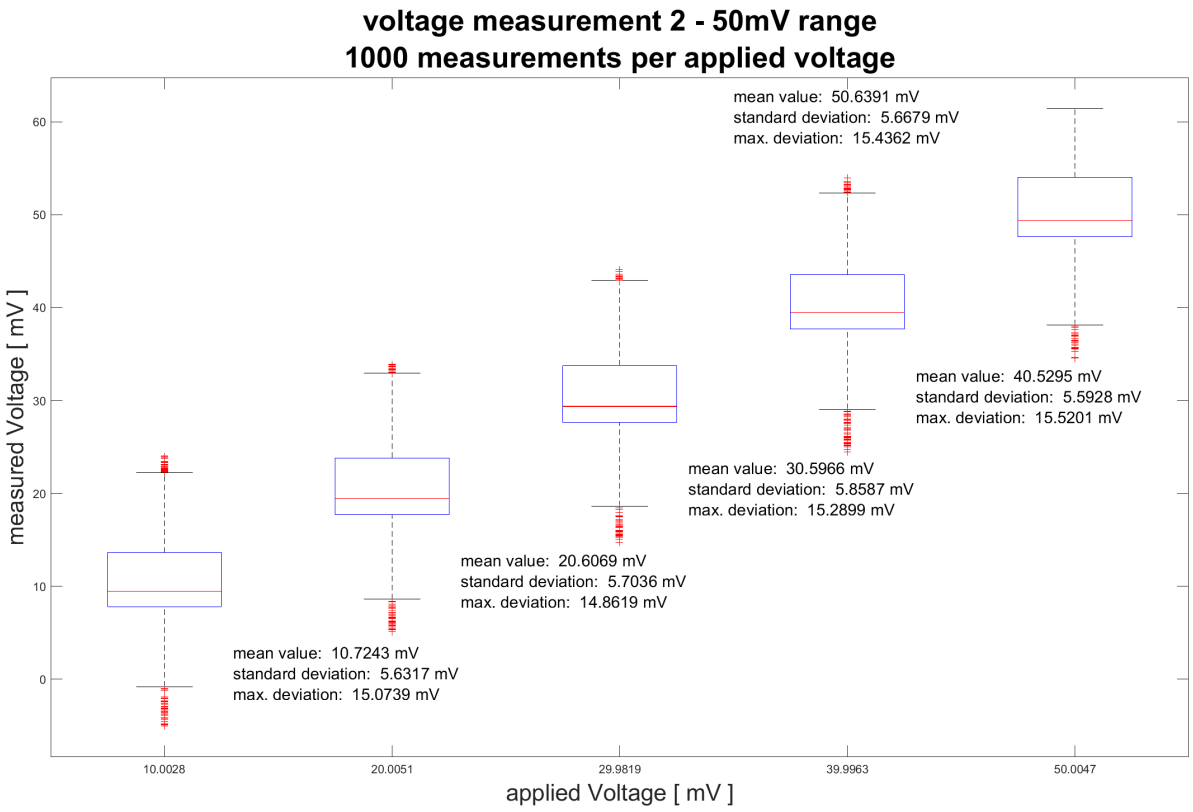


Figure 7.11: Voltage Measurement 2 - 50 mV range

7.3 Current Measurement

We did the same procedure with the current measurements. So we did 1000 measurements of the same current applied with Precision Supply Keysight B2912A and calculated the mean value and deviation of the measurements as you can see in the following plots.

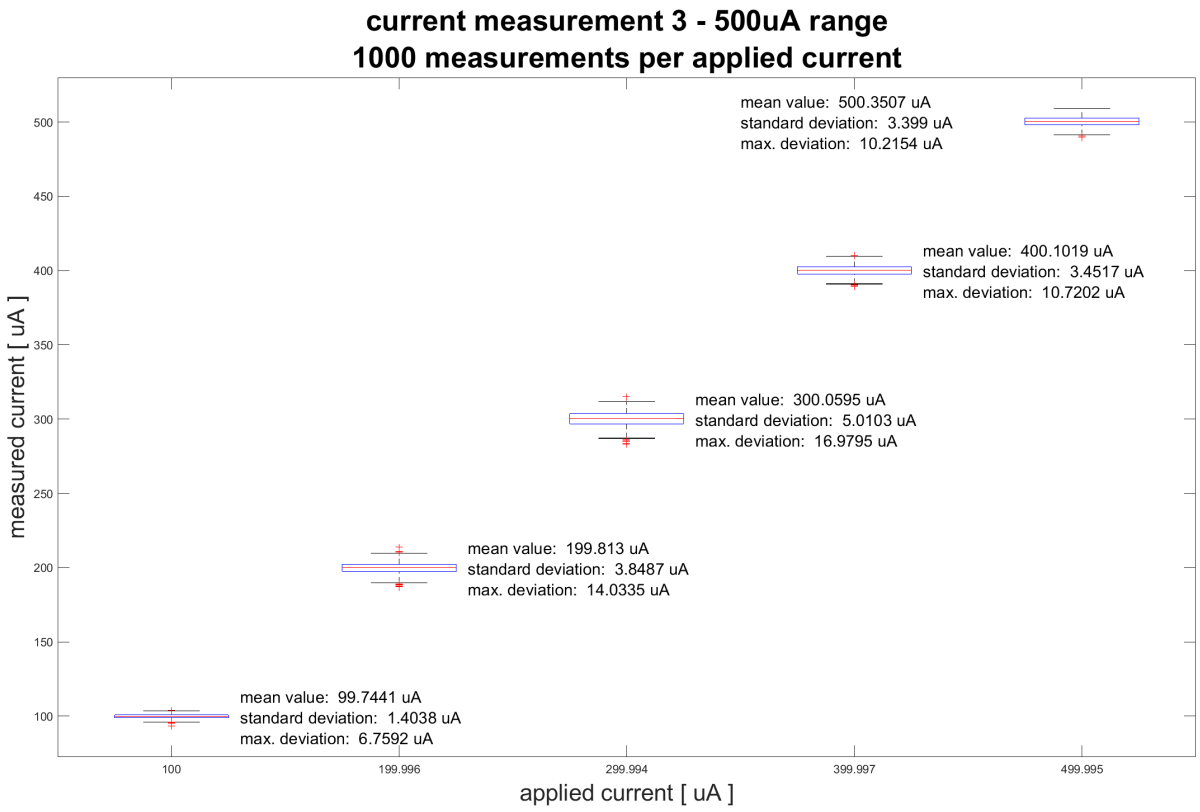


Figure 7.12: Current Measurement 3 - 500 µA range

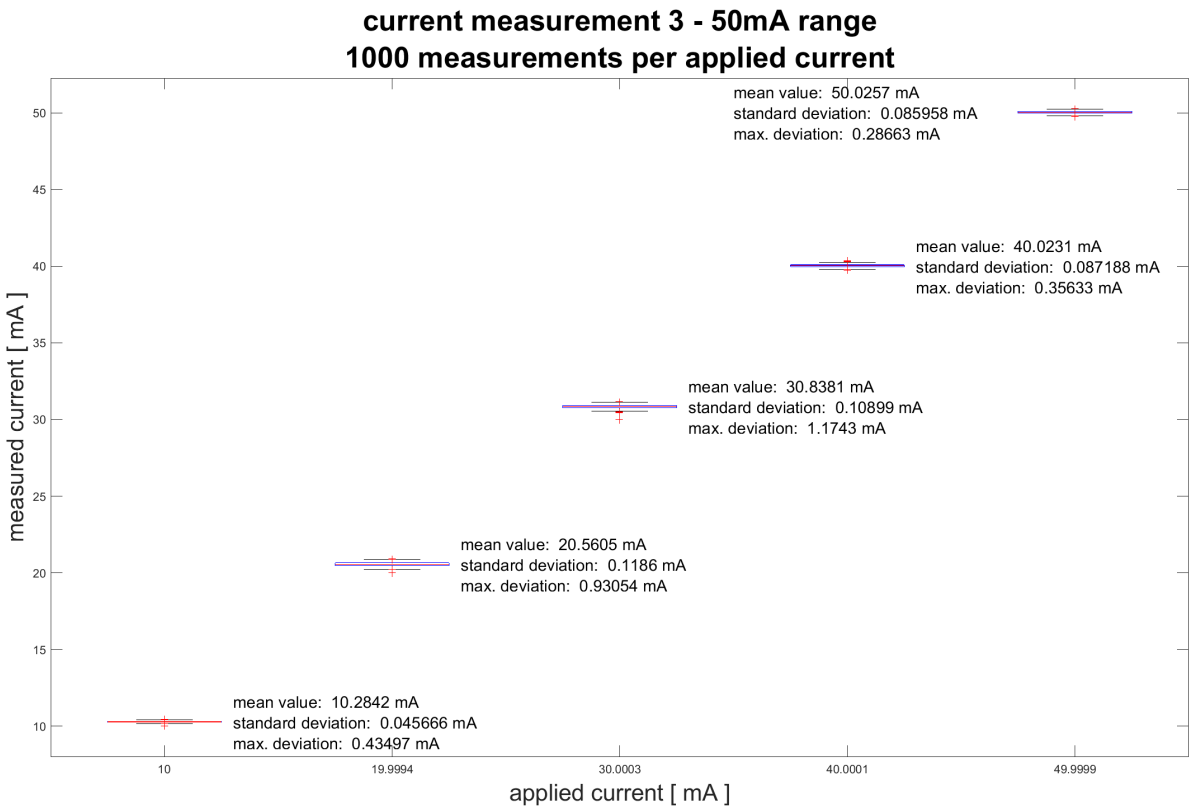


Figure 7.13: Current Measurement 3 - 50 mA range

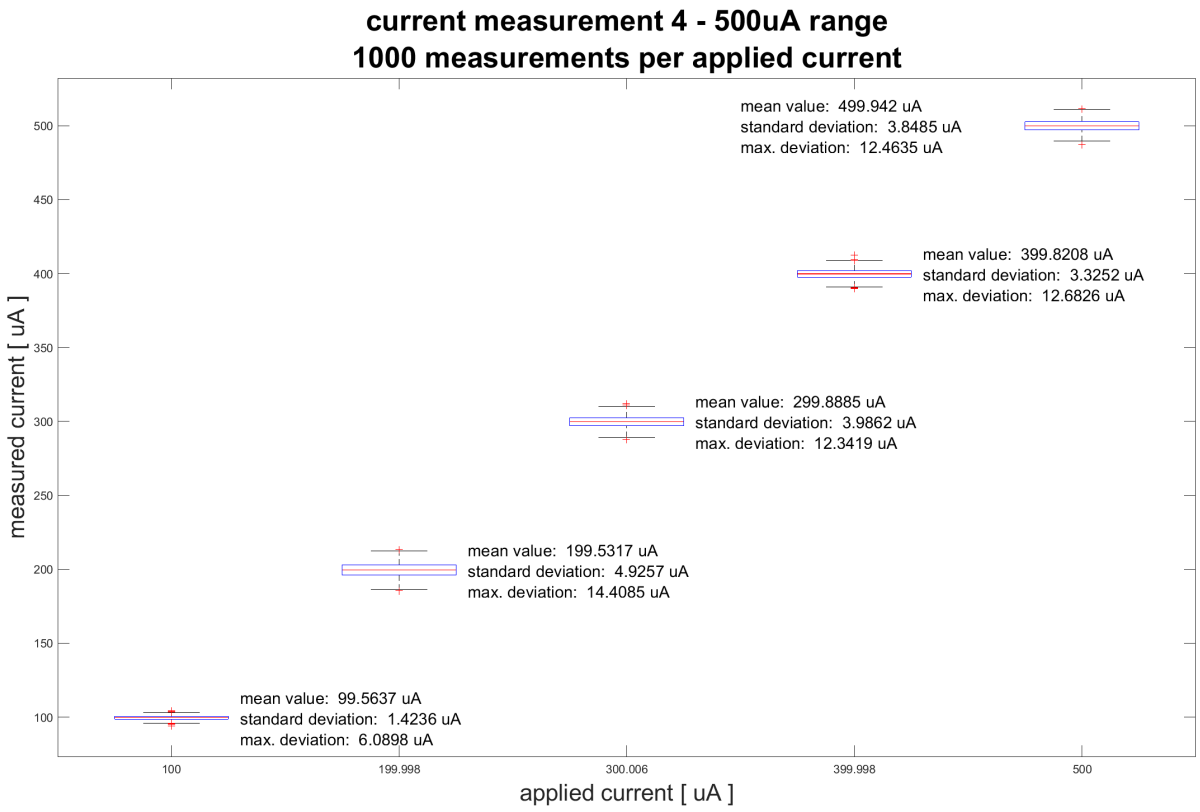


Figure 7.14: Current Measurement 4 - 500 µA range

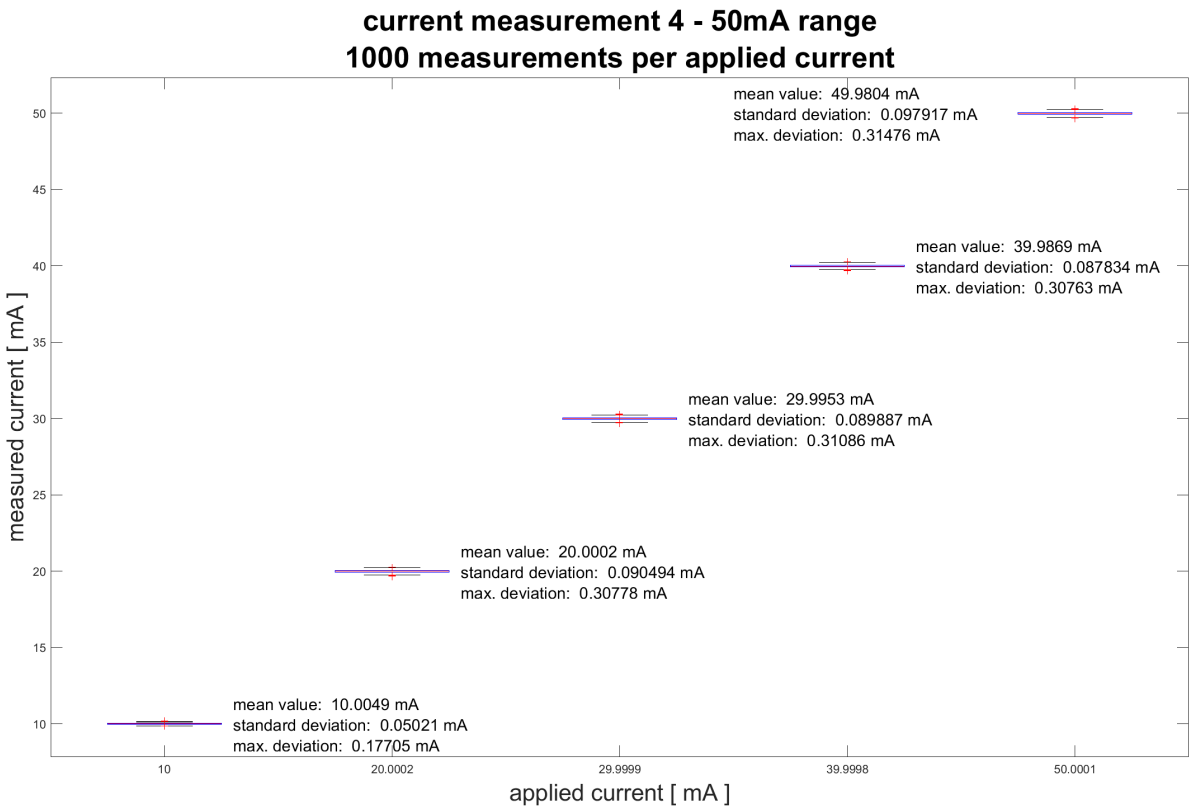


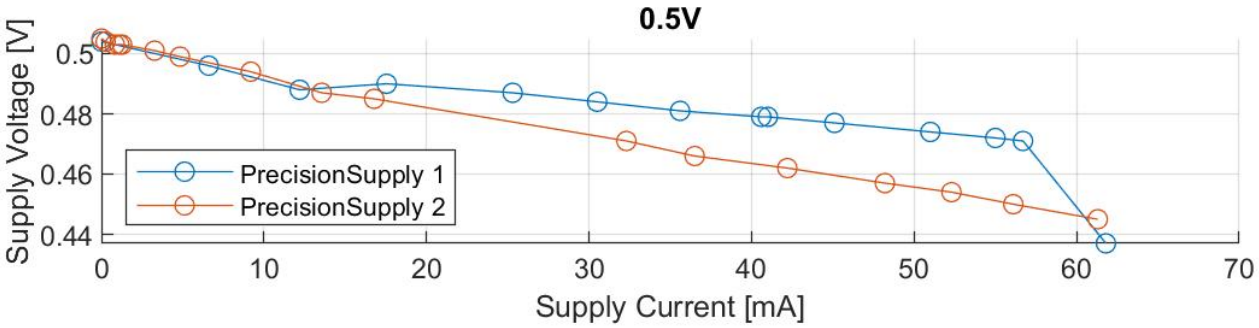
Figure 7.15: Current Measurement 4 - 50 mA range

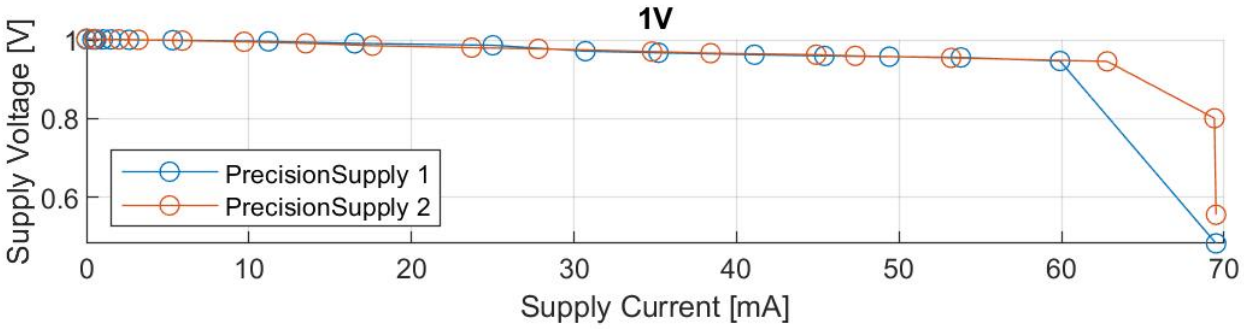
7.4 Supply Voltages

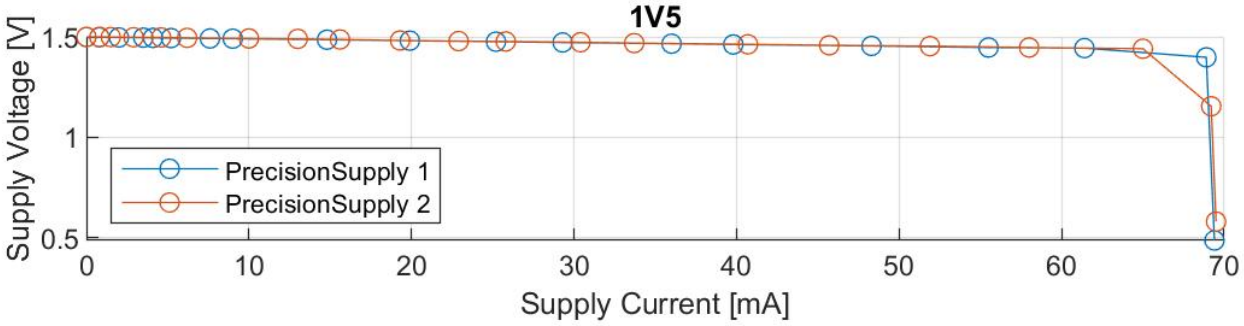
As already mentioned, the Demonstrator can generate different voltages. There are three so called Variable Supplies that generate voltages from 1.8 V to 3.3 V, there are two Precision Supplies which generate voltages from 0 V to 2.5 V and there are two Body Bias that generate Voltages from 0 to 2.5 V and from −2.5 V to 0 V. The Precision Supplies are the most accurate and voltages are generated with a precision of 0.005 V as well as the Body Bias voltages. The Variable Voltages are generated with a precision of 0.1 V. Those precisions are verified with the Digital Multimeter PeakTech 2005.

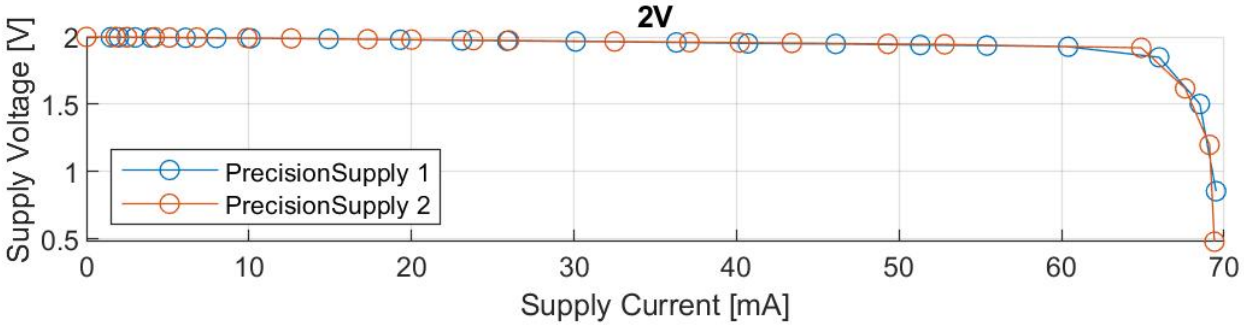
7.4.1 Changing The Load Resistance

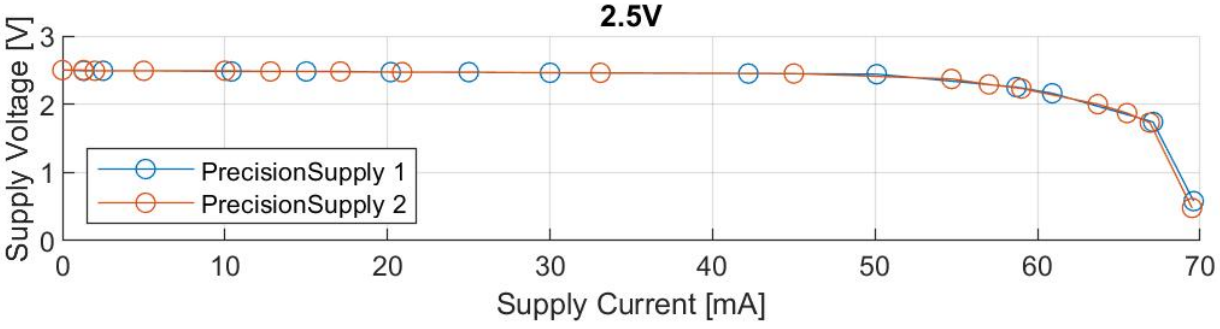
In the following plots one can see how the output voltages delivered from the Demonstrator behave as the load resistance is increased. As expected all the voltages decrease with an increasing output current. We measured those voltages with the Digital Multimeter PeakTech 2005.


0.5V
Supply Voltage [V]
0.5
0.48
0.46
0.44
PrecisionSupply 1
PrecisionSupply 2
0
10
20
30
40
50
60
70
Supply Current [mA]


1V
Supply Voltage [V]
1
0.8
0.6
PrecisionSupply 1
PrecisionSupply 2
0
10
20
30
40
50
60
70
Supply Current [mA]


1V5
Supply Voltage [V]
1.5
1
0.5
PrecisionSupply 1
PrecisionSupply 2
0
10
20
30
40
50
60
70
Supply Current [mA]


2V
Supply Voltage [V]
2
1.5
1
0.5
PrecisionSupply 1
PrecisionSupply 2
0
10
20
30
40
50
60
70
Supply Current [mA]


2.5V
Supply Voltage [V]
3
2
1
0
PrecisionSupply 1
PrecisionSupply 2
0
10
20
30
40
50
60
70
Supply Current [mA]

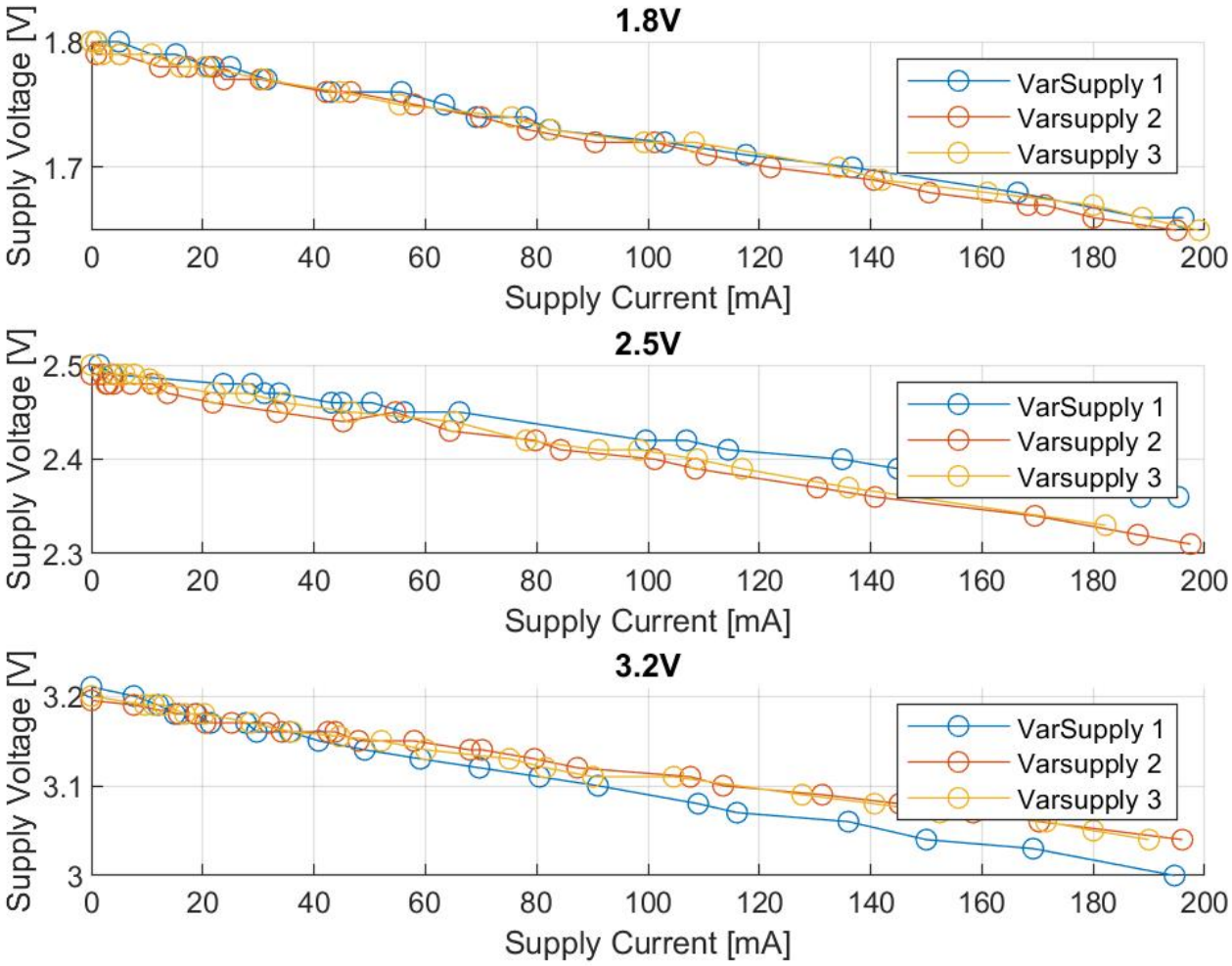


Figure 7.17: Voltage drop Variable Supply

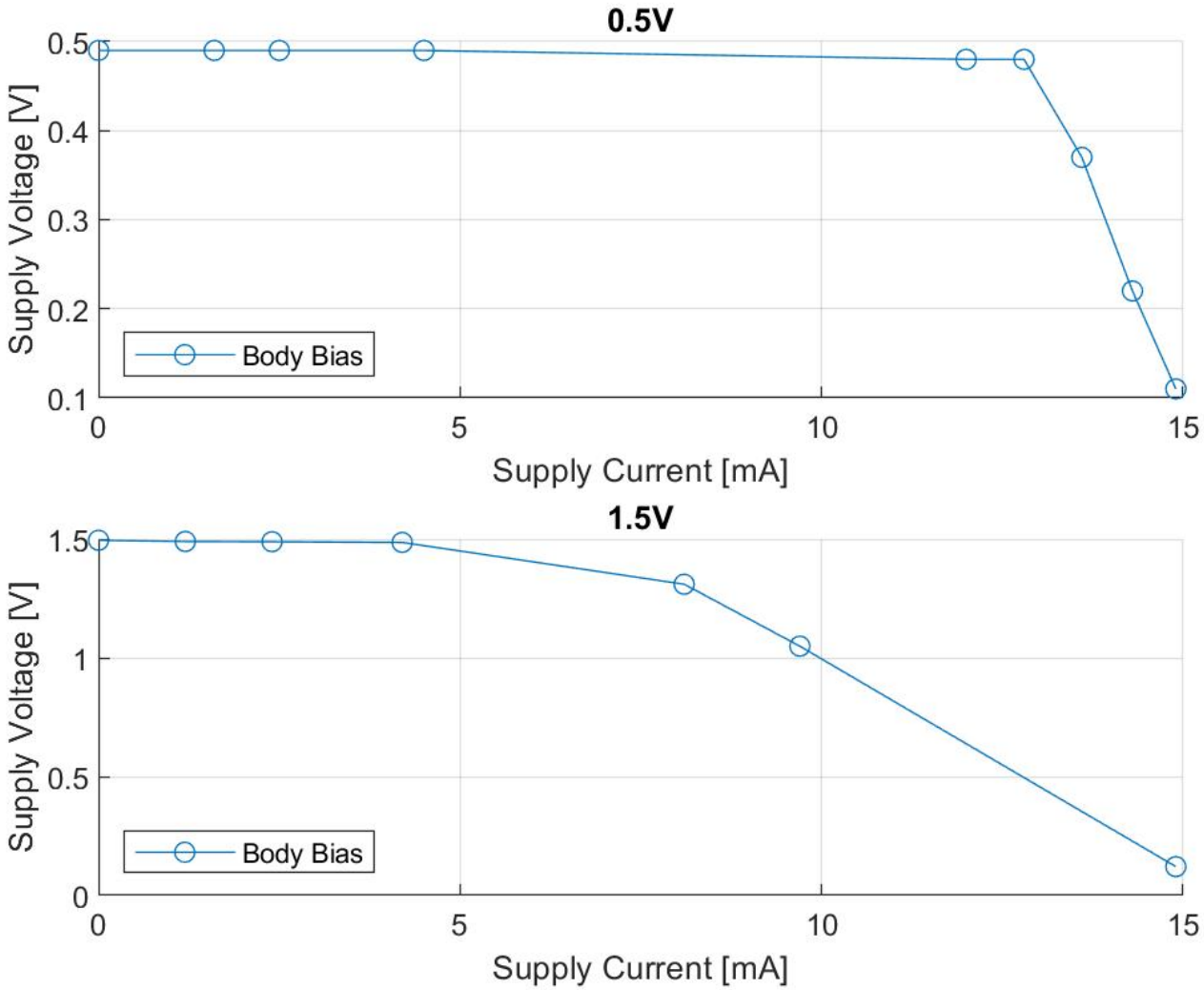


Figure 7.18: Voltage drop Body Bias

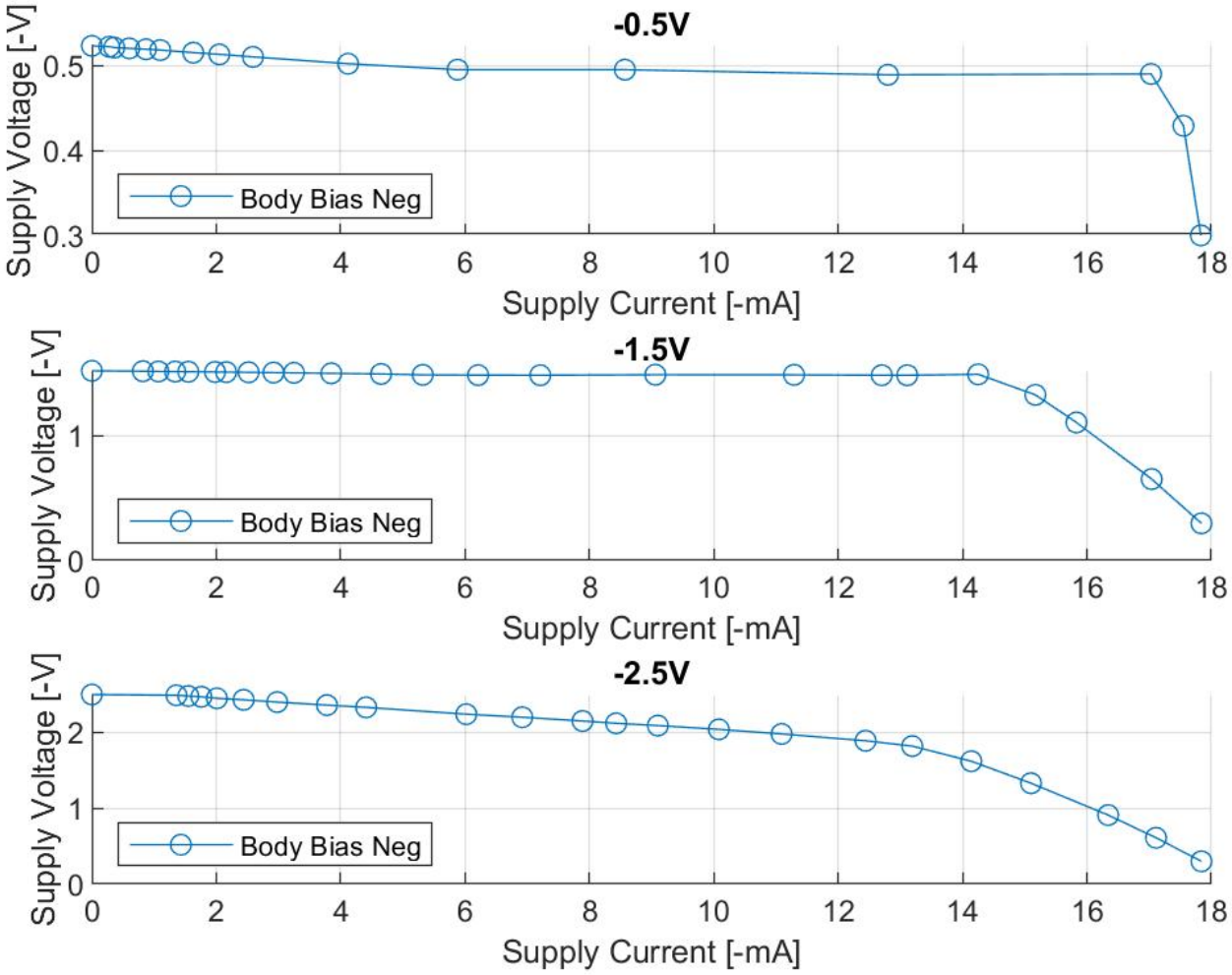
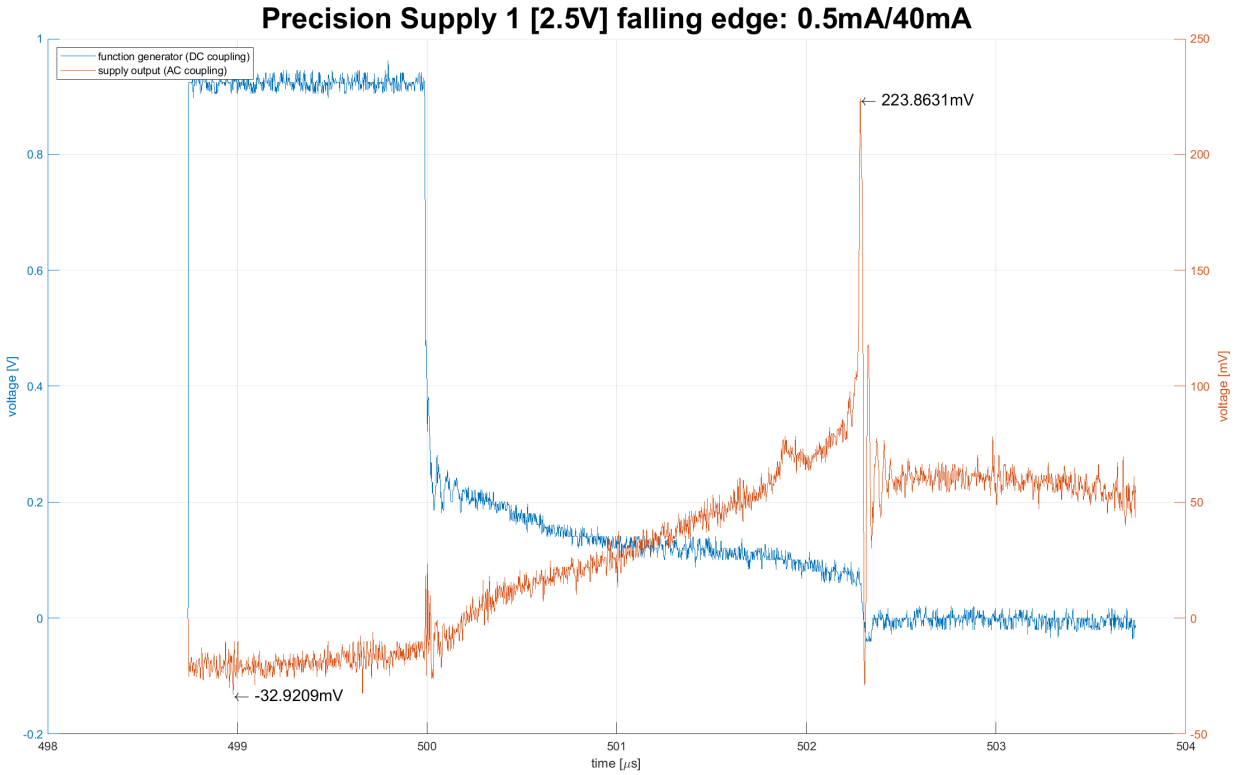
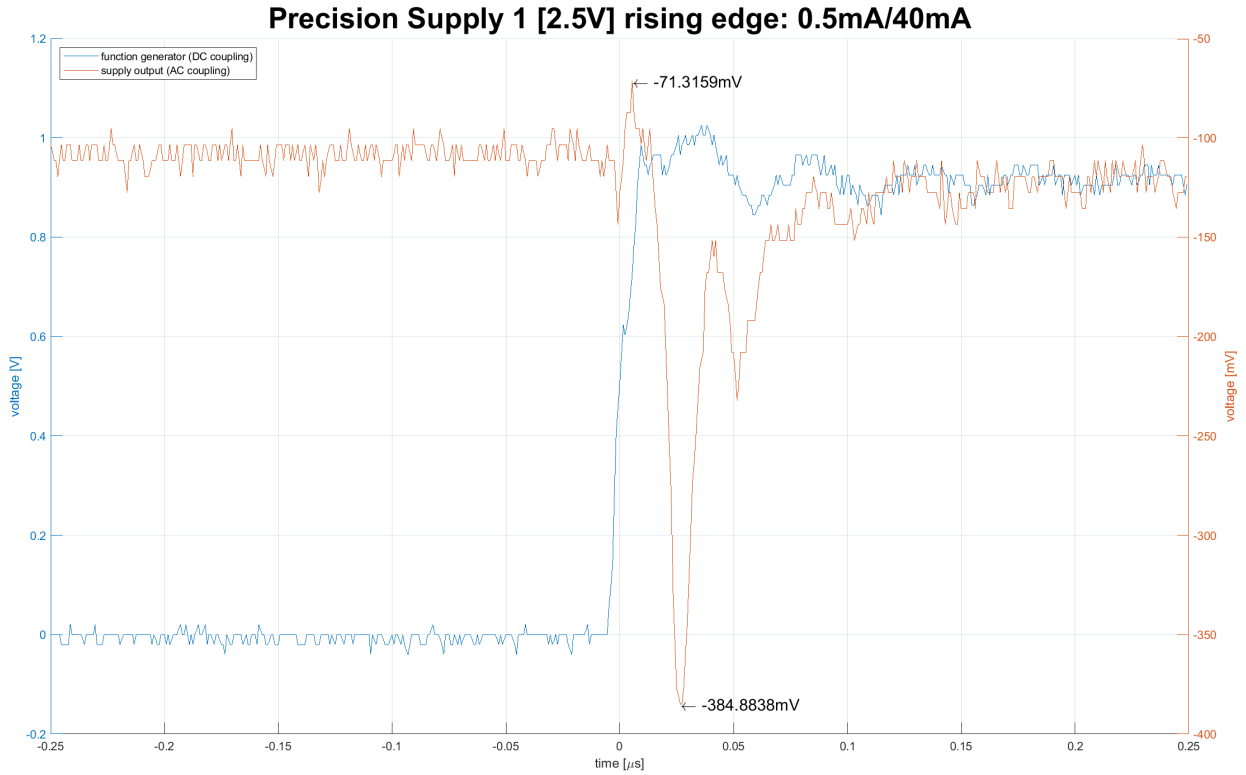


Figure 7.19: Voltage drop Negative Body Bias

7.4.2 Abruptly changing Load Current

Another important characteristic considering the output voltages is how they behave when the load currents change abruptly from one value to another. To switch between the different currents we connected the collector of the transistor MPSA 42 to the output of the Demonstrator with a potentiometer in between. With the square output voltage of the Keysight Waveform Generator 33600A Series at the base of the transistor we can switch it on and off. To regulate the current we simply changed the resistor of the potentiometer. Additionaly we connected a potentiometer in parallel to regulate the current when the transistor is closed. So for example we switched from 0.5 mA to 40 mA output current of the precision supply. In the following graphs the blue line represents output of the function generator measured with the Oscilloscope Keysight InfiniiVision MSOX2024A. When the output of the function generator is high, more current flows and vice versa. As we zoom into a falling or a rising edge of the output one can see that the supply voltage, which is represented by the orange line, oscillates first before it stabilizes again. The output voltage is AC coupled to measure the voltage difference, while the function generator is measured with DC coupling. In the title of the figures you can see the two currents and the output voltage point of the supply.


Precision Supply 1 [2.5V] falling edge: 0.5mA/40mA
function generator (DC coupling)
supply output (AC coupling)
223.8631mV
-32.9209mV
voltage [V]
voltage [mV]
time [μs]
1
0.8
0.6
0.4
0.2
0
-0.2
250
200
150
100
50
0
-50
498
499
500
501
502
503
504


Precision Supply 1 [2.5V] rising edge: 0.5mA/40mA
function generator (DC coupling)
supply output (AC coupling)
-71.3159mV
-384.8838mV
voltage [V]
voltage [mV]
time [μs]
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-50
-100
-150
-200
-250
-300
-350
-400
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
0.25

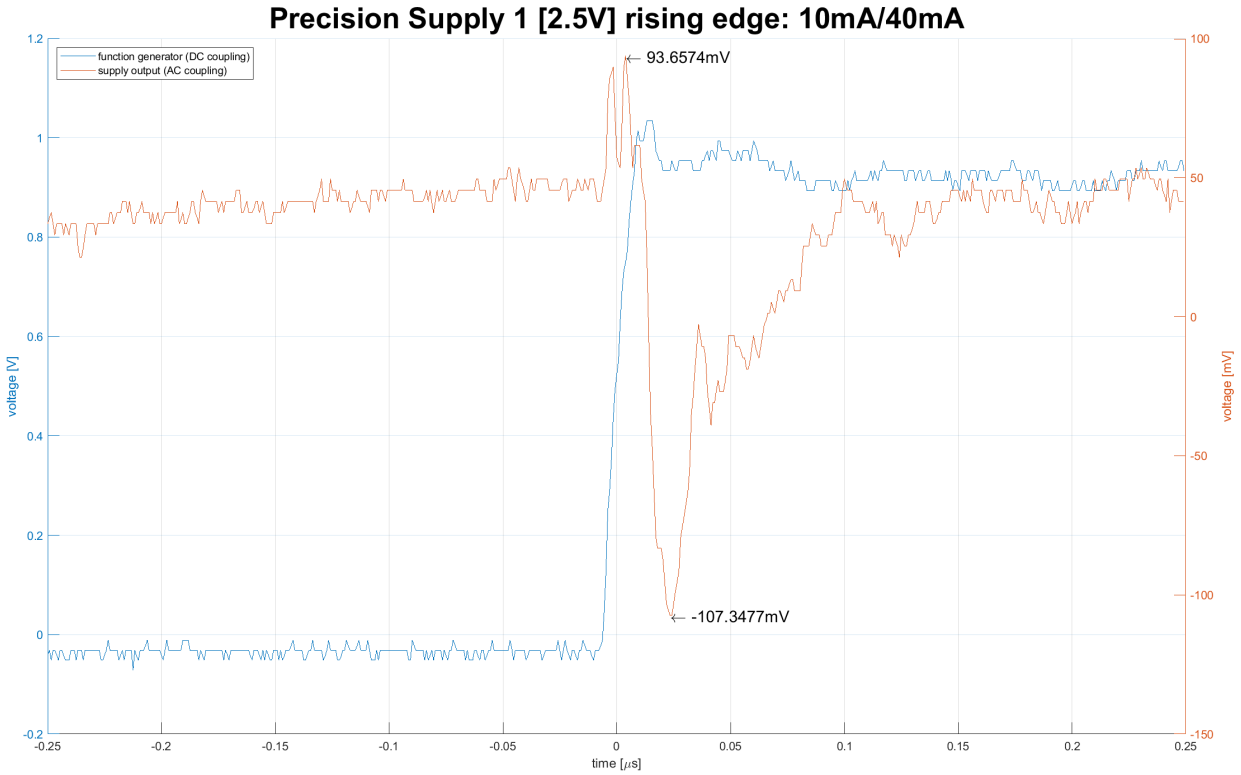
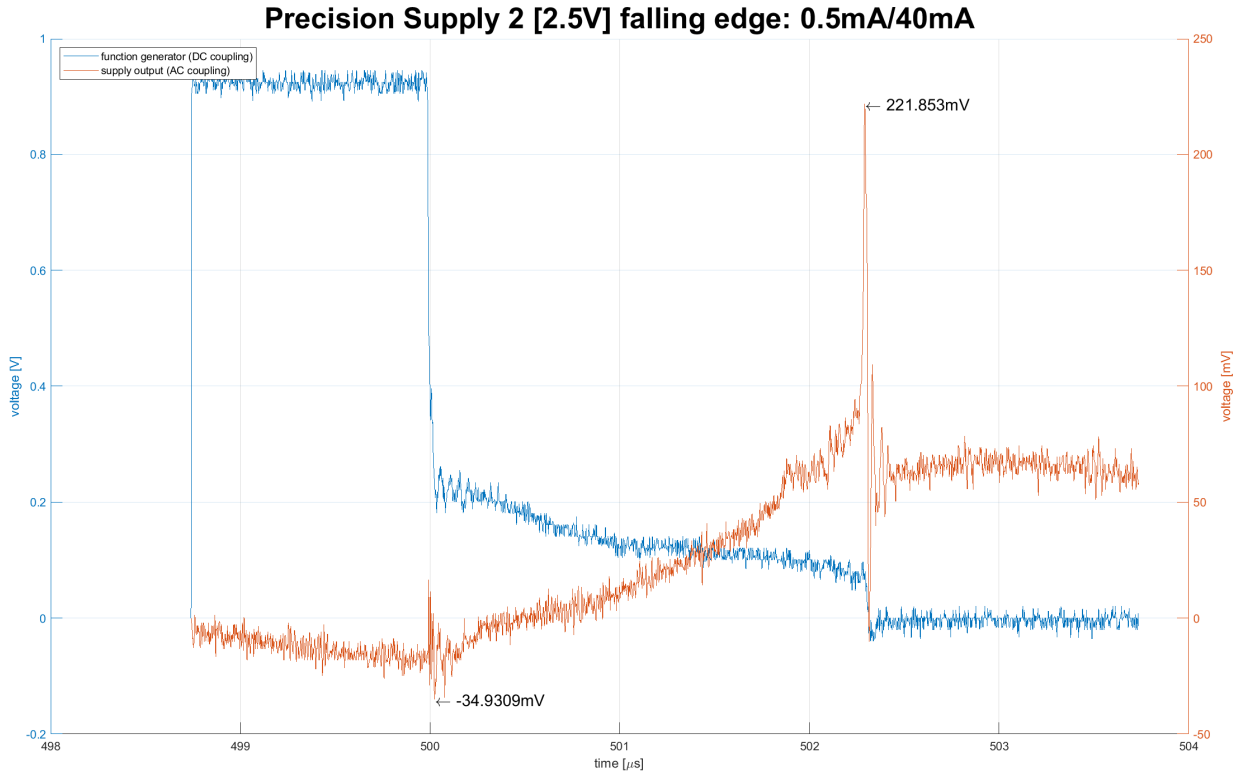
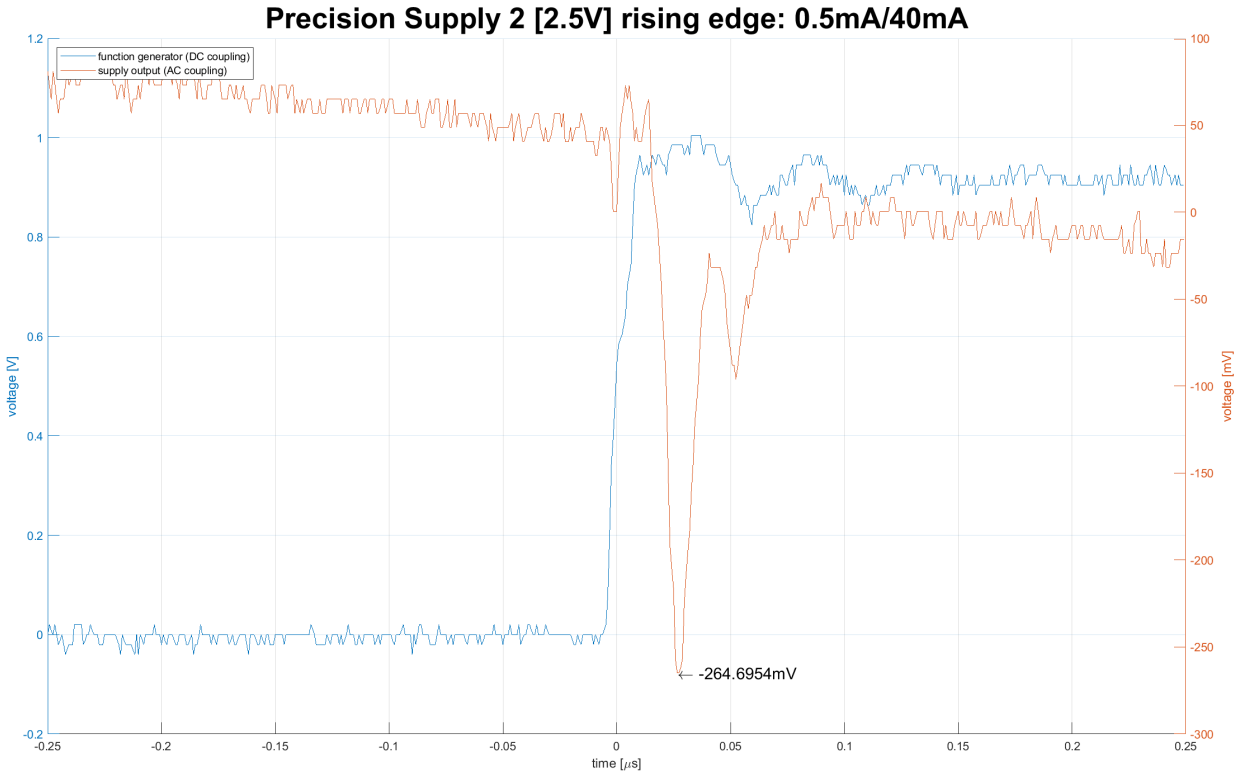


Figure 7.20: Load response precision supply 1.


Precision Supply 2 [2.5V] falling edge: 0.5mA/40mA
function generator (DC coupling)
supply output (AC coupling)
← 221.853mV
← -34.9309mV
voltage [V]
voltage [mV]
time [μs]
1
0.8
0.6
0.4
0.2
0
-0.2
250
200
150
100
50
0
-50
498
499
500
501
502
503
504


Precision Supply 2 [2.5V] rising edge: 0.5mA/40mA
function generator (DC coupling)
supply output (AC coupling)
voltage [V]
voltage [mV]
time [μs]
-264.6954mV
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
100
50
0
-50
-100
-150
-200
-250
-300
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
0.25

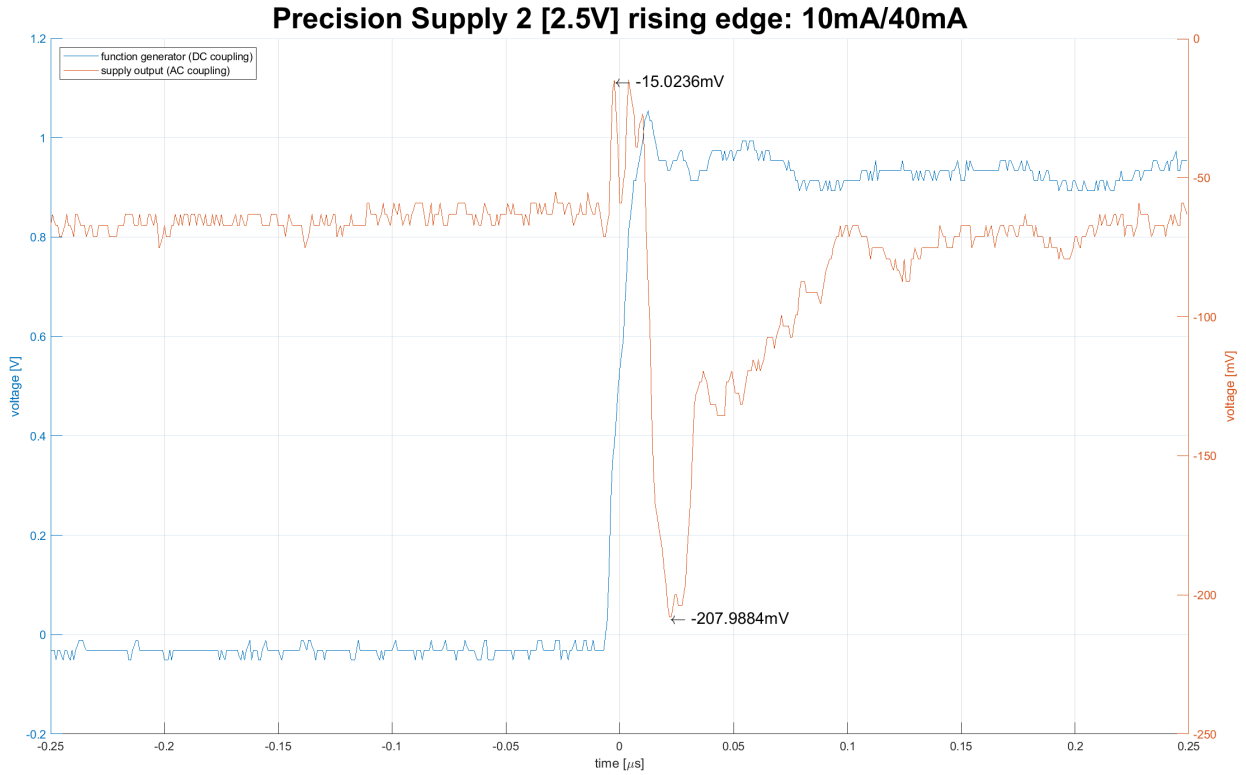
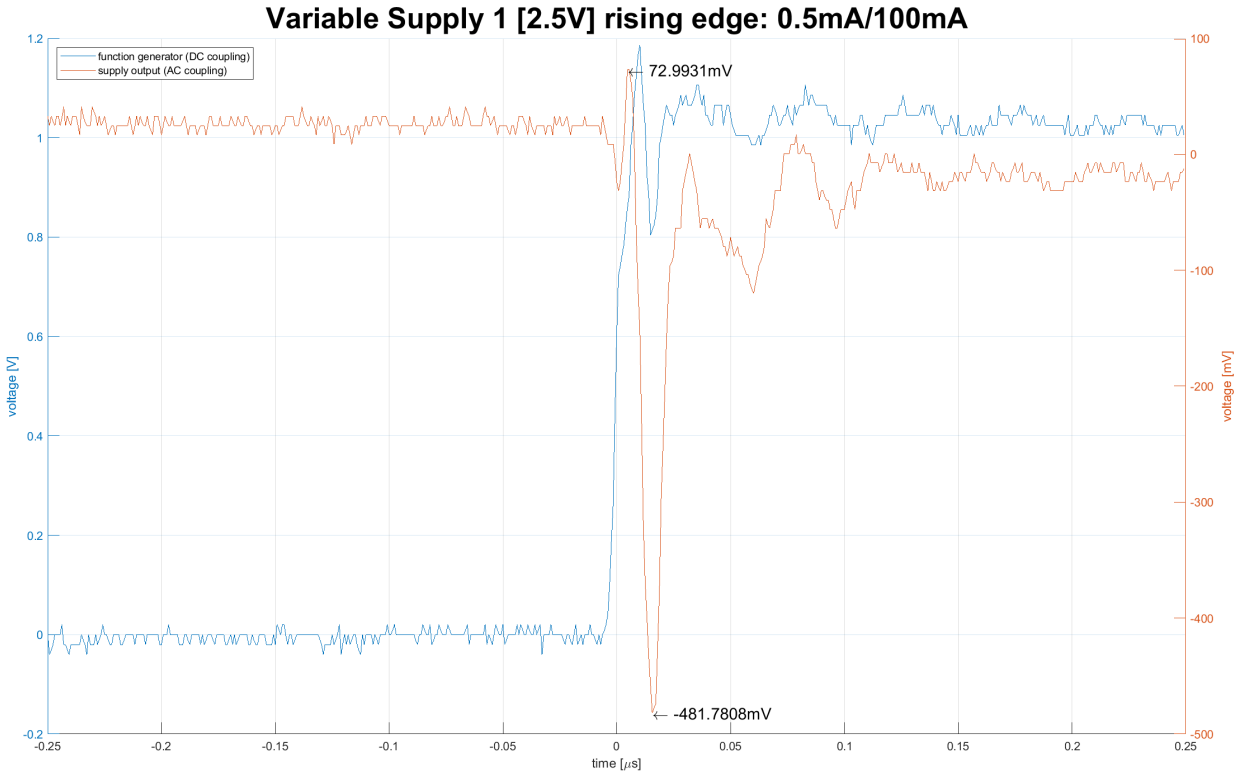


Figure 7.21: Load response precision supply 2.


Variable Supply 1 [2.5V] rising edge: 0.5mA/100mA
function generator (DC coupling)
supply output (AC coupling)
72.9931mV
-481.7808mV
voltage [V]
voltage [mV]
time [μs]
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
100
0
-100
-200
-300
-400
-500
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
0.25

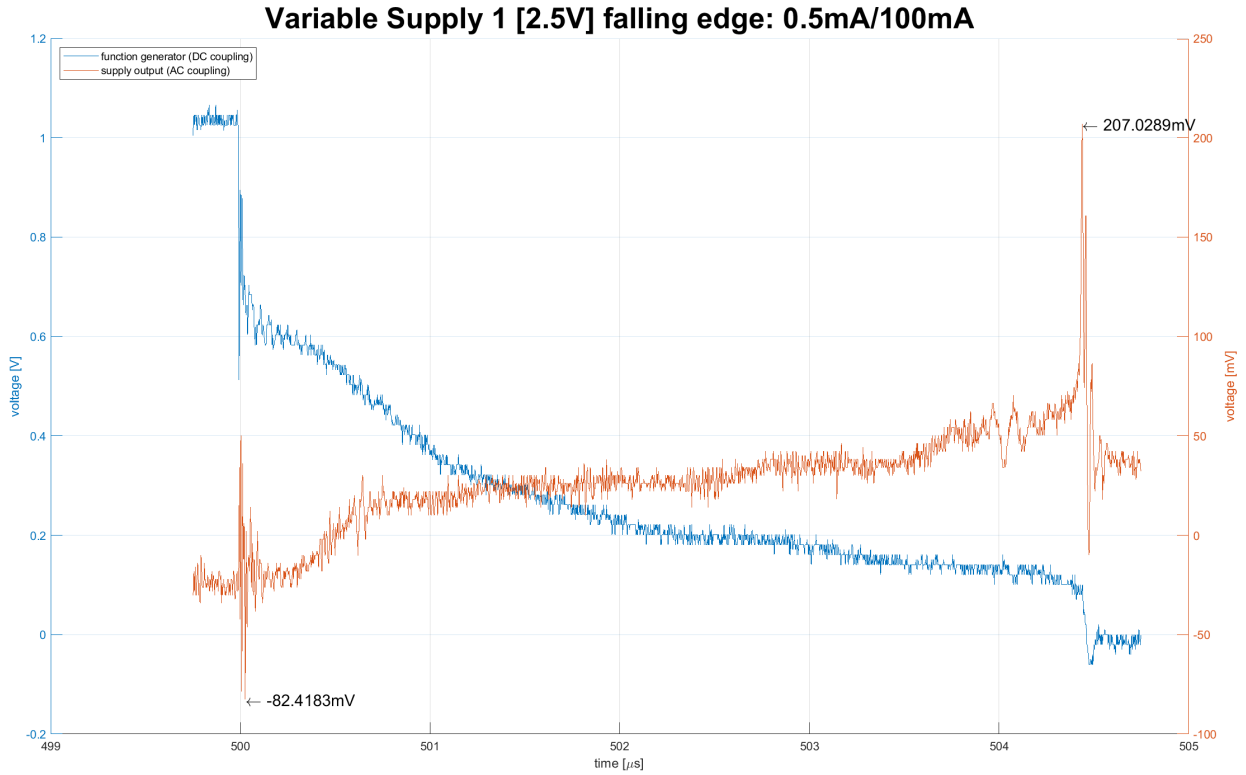
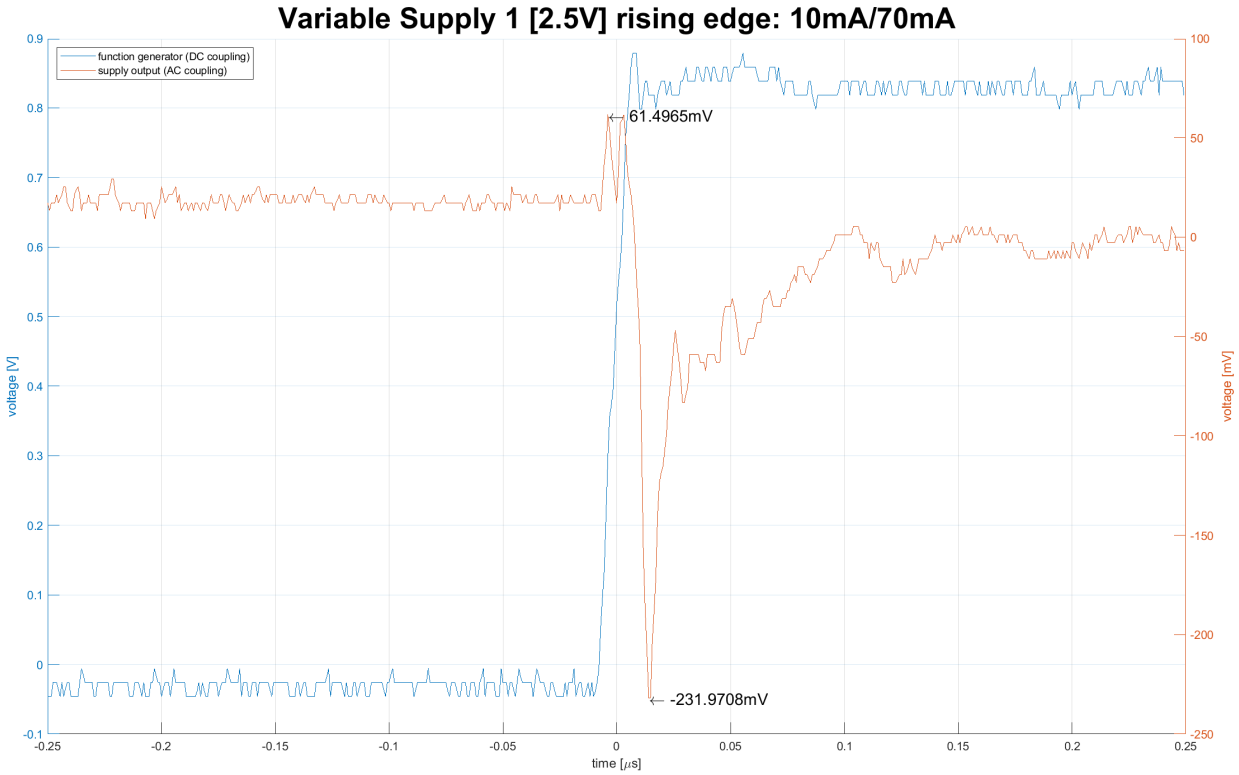


Figure 7.22: Load response variable supply 1: 50mA to 100mA.


Variable Supply 1 [2.5V] rising edge: 10mA/70mA
function generator (DC coupling)
supply output (AC coupling)
61.4965mV
-231.9708mV
voltage [V]
voltage [mV]
time [μs]
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
100
50
0
-50
-100
-150
-200
-250
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
0.25

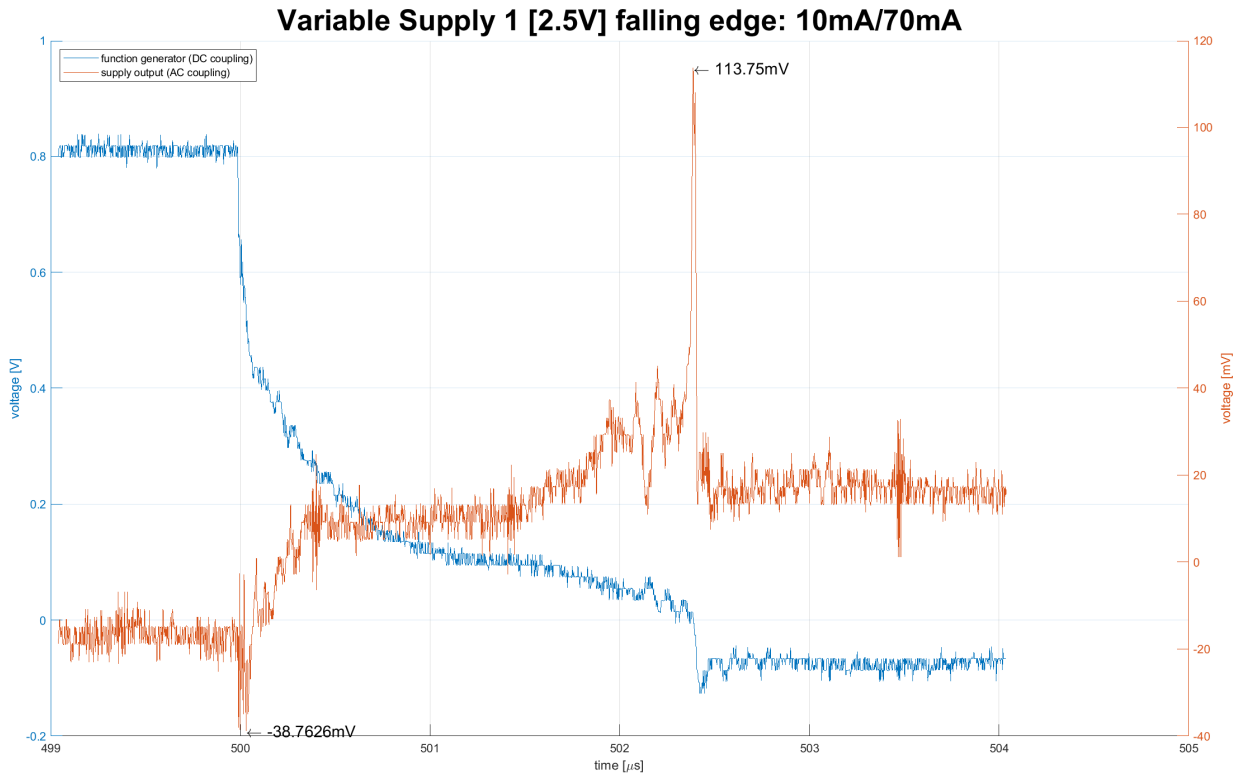
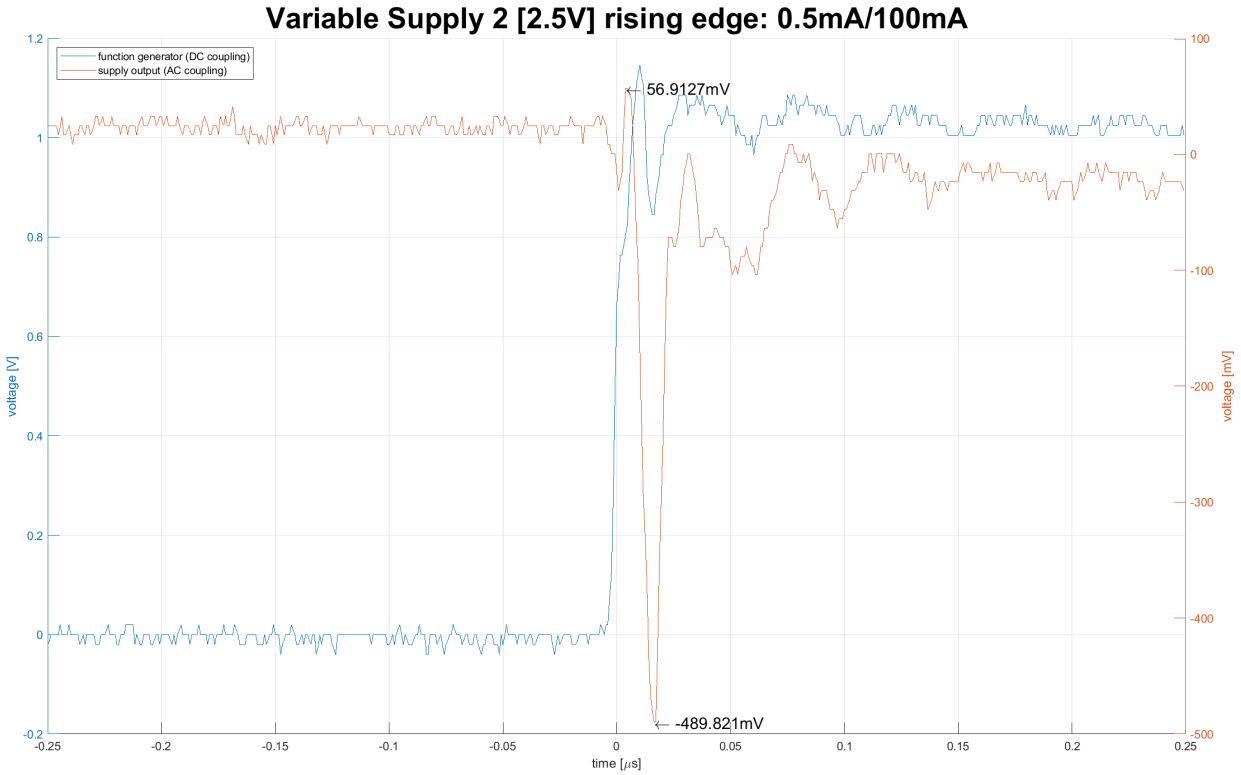


Figure 7.23: Load response variable supply 1: 10mA to 70mA.


Variable Supply 2 [2.5V] rising edge: 0.5mA/100mA
function generator (DC coupling)
supply output (AC coupling)
56.9127mV
-489.821mV
voltage [V]
voltage [mV]
time [μs]
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
100
0
-100
-200
-300
-400
-500
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
0.25

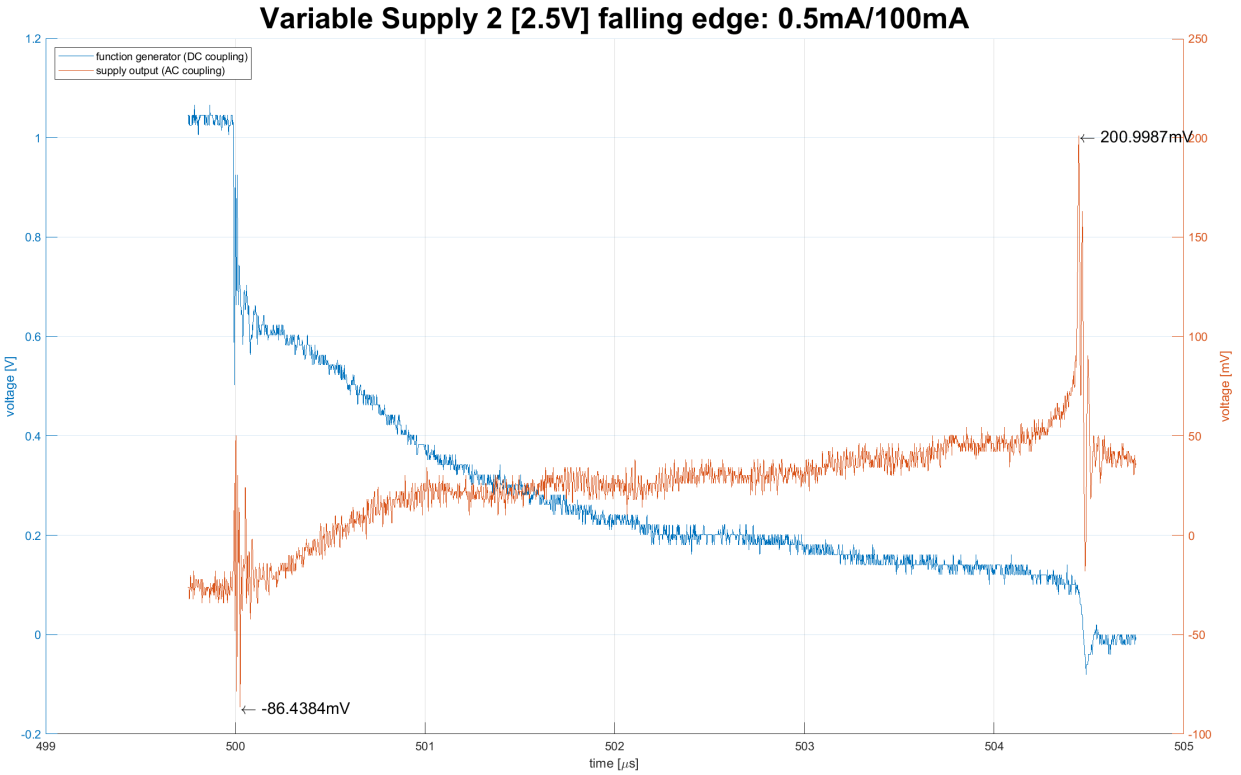
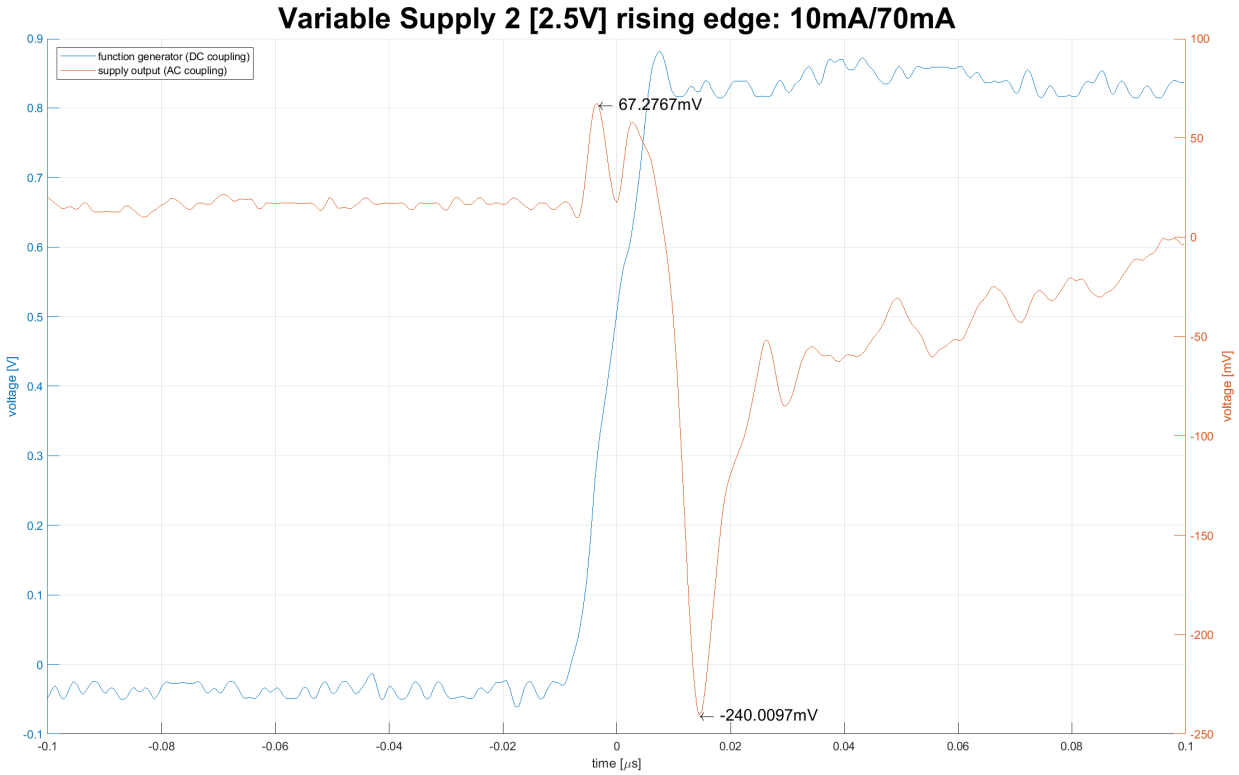


Figure 7.24: Load response variable supply 2: 0.5mA to 100mA.


Variable Supply 2 [2.5V] rising edge: 10mA/70mA
function generator (DC coupling)
supply output (AC coupling)
67.2767mV
-240.0097mV
voltage [V]
voltage [mV]
time [μs]
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
100
50
0
-50
-100
-150
-200
-250
-0.1
-0.08
-0.06
-0.04
-0.02
0
0.02
0.04
0.06
0.08
0.1

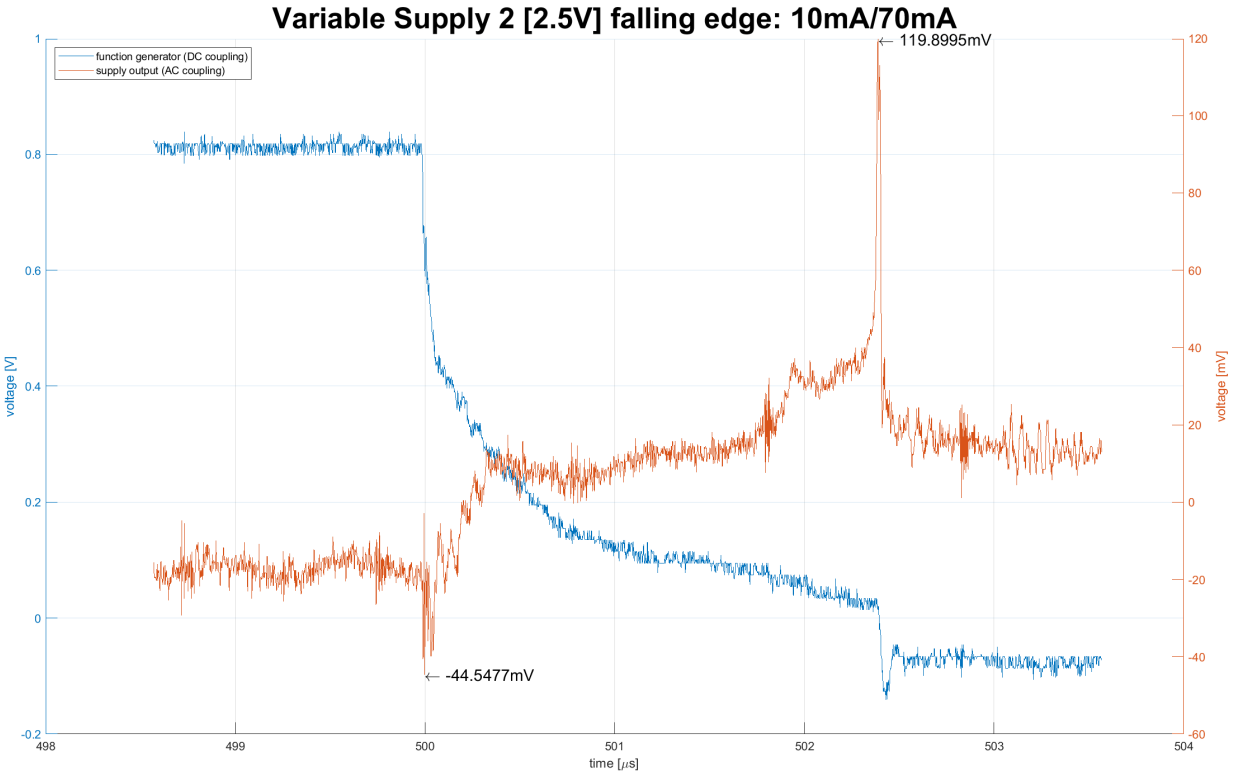
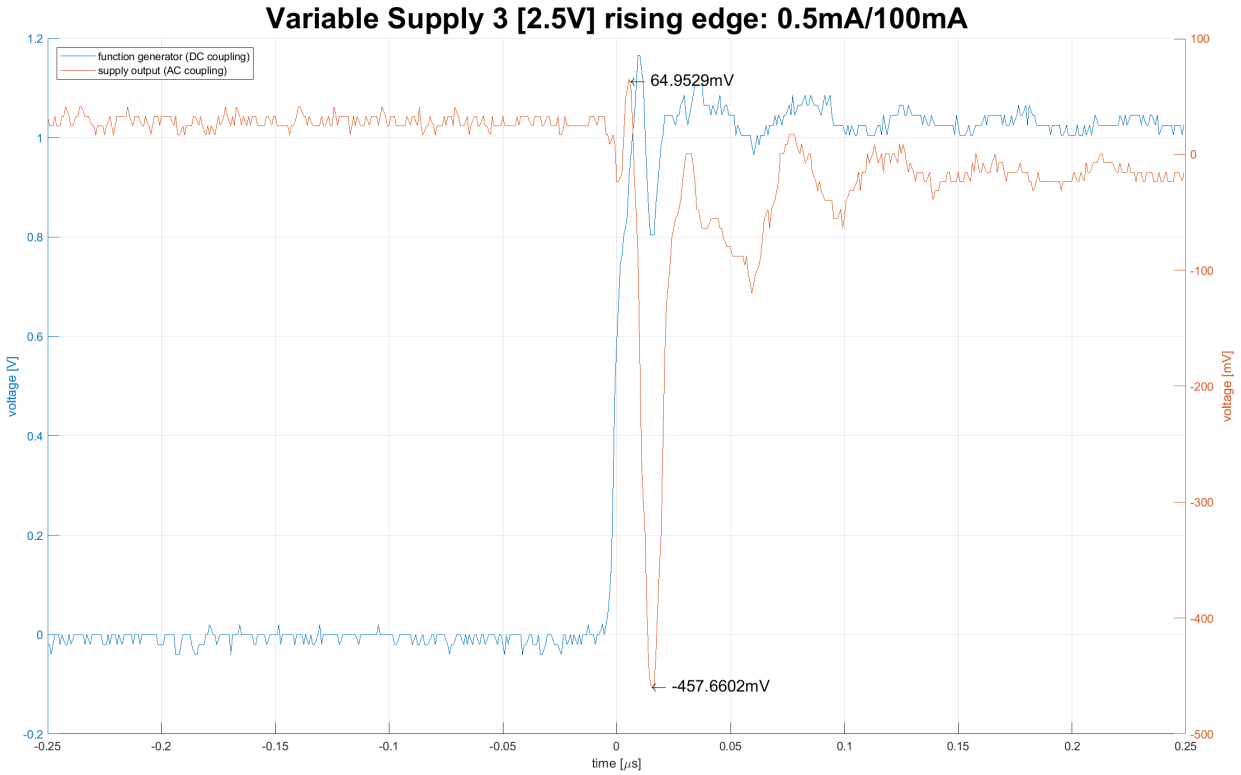


Figure 7.25: Load response variable supply 2: 10mA to 70mA.


Variable Supply 3 [2.5V] rising edge: 0.5mA/100mA
function generator (DC coupling)
supply output (AC coupling)
64.9529mV
-457.6602mV
voltage [V]
voltage [mV]
time [μs]
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
100
0
-100
-200
-300
-400
-500
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
0.25

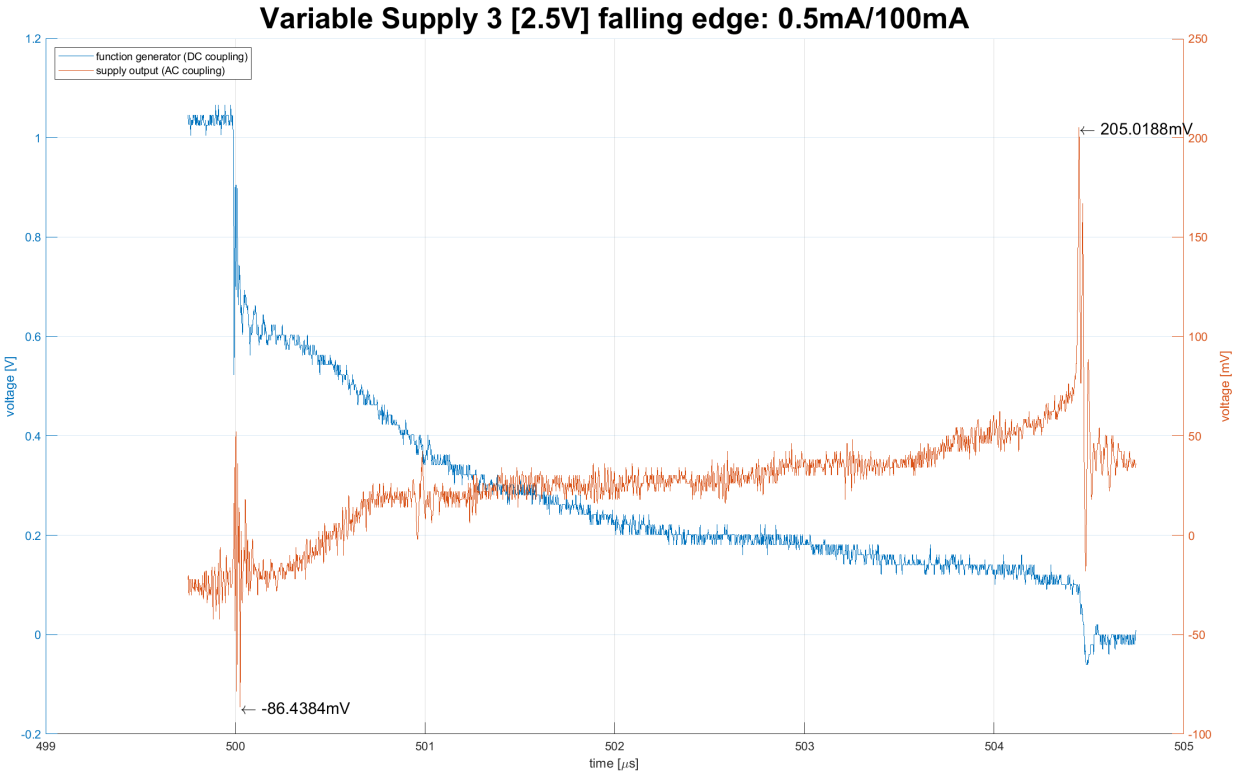
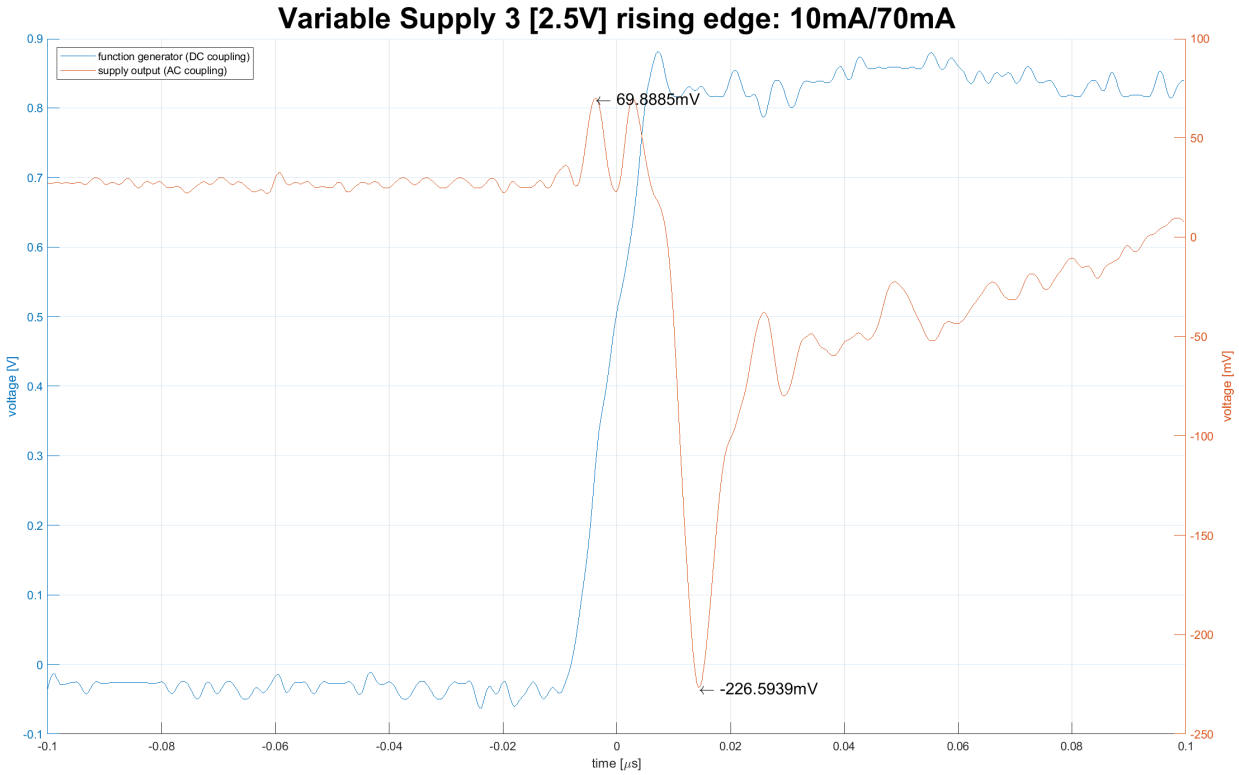


Figure 7.26: Load response variable supply 3: 0.5mA to 100mA.


Variable Supply 3 [2.5V] rising edge: 10mA/70mA
function generator (DC coupling)
supply output (AC coupling)
69.8885mV
-226.5939mV
voltage [V]
voltage [mV]
time [μs]
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
100
50
0
-50
-100
-150
-200
-250
-0.1
-0.08
-0.06
-0.04
-0.02
0
0.02
0.04
0.06
0.08
0.1

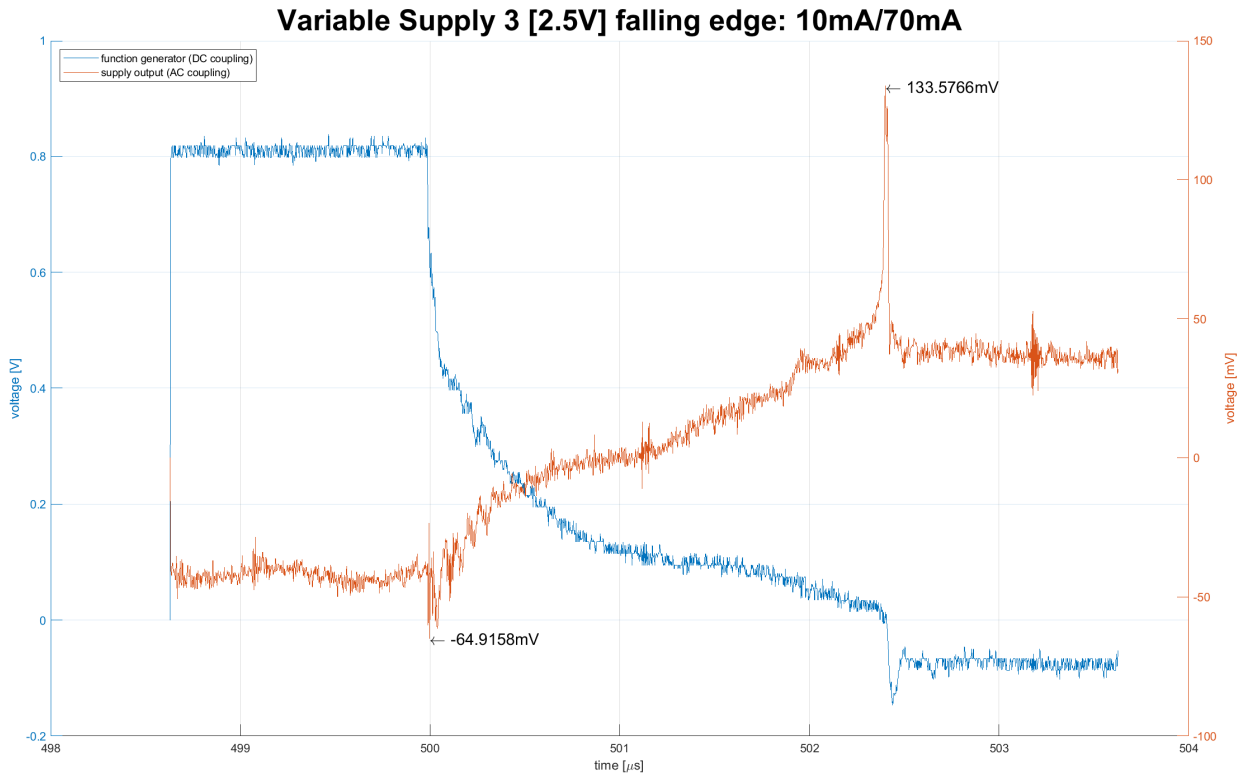


Figure 7.27: Load response variable supply 3: 10mA to 70mA.

Chapter 8

Errata

8.0.1 Variable Supplies

The datasheet of the TPS62742 says: "Once the input voltage comes close to the output voltage, the DC/DC converter stops switching and enters 100% duty cycle operation. It connects the output VOUT via the inductor and the internal high side MOSFET switch to the input VIN, ...".
We could find neither a number nor a drawing describing "input close to output". Thus, if you want to set the output voltage to 3.3 V it switches to 3.5 V. This could be due to the supply voltage of 3.5 V being too close to 3.3 V[1].

8.0.2 ADS8885

We connected the DIN of the four ADS8885 to the MOSI of the Raspberry Pi, like in the single chip configuration. But as seen in figure 56 in the datasheet the Chip Select (CS) of the Raspberry Pi should be connected to the DIN of the four ADS8885. That's why we had to correct the error by software, which leads to a loss of performance. The solution is to set the GPIO pins manually and don't use the hardware SPI library[2].

8.0.3 Daughter Board

The COM net of the voltage and current measurement has the potential 1.35 V, so it can't be connected to the AGND. It draws too much current and the voltage drops to 0 V of the reference voltage. Therefore, the AGND must remain floating.

8.0.4 Body Bias

Unfortunately, the Body Bias output reaches only 1.93 V, although earlier we could reach our desired 2.5 V. The problem is the LM321MF, since the output voltage of the LTC2631 is correct for all voltages. We replaced the operation amplifier and all

[1] Datasheet: TIDA-01012
[2] Datasheet: LTC2631CTS8-LZ12#TRMPBF

surroundings, but we didn't detect any changes. Due to lack of time we couldn't find the bug.

8.0.5 PiJuice Header

After connecting a new battery to the Pi Juice Header we were not able to get accurate information about it's charge using the Pi Juice API.
In addition, it sometimes happens, that the Power On Button won't start up the Raspberry Pi unless it is connected to a power source while the button is pressed. We believe that this is due to a problem of the Pi Juice Header, since we configured everything according to the manual.